Contents

Contents

1 Spatial filters

The Fourier Transform is an indispensable tool for linear systems analysis, image analysis, and for efficient filter output computation. Among the benefits of the Fourier transform representation: it's easy to analyze images according to spatial frequency, and this represents some progress in interpreting the image over merely a pixel representation. But the Fourier transform has a major drawback as an image representation: it's too global! Every sinusoidal component covers the entire image. The Fourier transform tells us a little about *what* is happening in the image (based on the spatial frequency content), but it tells us nothing about *where* it is happening.

On the other hand, a pixel representation gives great spatial localization, but a pixel value by itself doesn't help us learn much about what's going on in the image. A Fourier representation tells us a bit about what's going on, but nothing about where it happens. We seek a representation that's somewhere in between those two extremes.

1.1 Filters

Linear convolutions, despite their simplicity, are surprisingly useful for processing and interpreting images. It's often very useful to blur images, in preparation for subsampling or to remove noise, for example. Other useful processing includes edge enhancement and motion analysis.

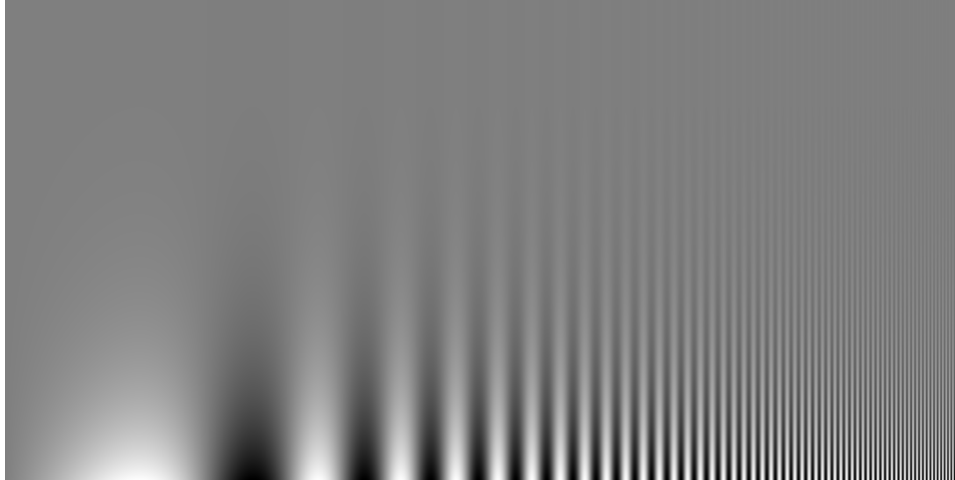From the previous section we know that we can write linear filters as convolutions:

$$f[n,m] = h[n,m] \circ g[n,m] \tag{1.1}$$

where $h[n,m]$ is the impulse response of the system. We can also write this as a product in the Fourier domain:

$$F[u,v] = H[u,v] G[u,v] \tag{1.2}$$

The function $H[u,v]$ is called the transfer function of the filter. If we use the polar form:

$$H[u,v] = |H[u,v]| \exp(j \angle H[u,v]) \tag{1.3}$$

Figure 1.1
Contrast sensitivity function shown by the Campbell & Robson chart.

The magnitude $|H[u, v]|$ is the amplitude gain, and the phase $\angle H[u, v]$ is the phase shift.

The Fourier domain shows that, in many cases, what a filter does is to block or let pass certain frequencies. Filters are many times classified according to the frequencies that they let pass through the filter (low, medium or high frequencies):

- Low-pass filter
- Band-pass filter
- High-pass filter

But this classification of a filter might not be appropriate in many cases. Some filters have their main effect over the phase of the signal and they are better understood in the spatial domain. In general, filters affect both the magnitude and the phase of the input signal.

In this chapter we will analyze several important linear filters. We will study spatial filters and then temporal filters.

1.2   Human visual system and the contrast sensitivity function

Before we start describing different types of linear filters, let's start by gaining some subjective experience by playing with one: our own visual system. Although our visual system is clearly a non-linear system, linear filter theory can explain some aspects of our perception. Indeed, under certain conditions, the first stages of visual computation perform of the visual system can be approximated by a linear filter.

To experience the transfer function of our own visual system, let's build the following $N \times M$ image:

$$\mathbf{I}[n, m] = A[m] \sin(2\pi f[n] n/N) \tag{1.4}$$

with

$$A[m] = A_{min} \left(\frac{A_{max}}{A_{min}} \right)^{m/M} \tag{1.5}$$

and

$$f[n] = f_{min} \left(\frac{f_{max}}{f_{min}} \right)^{n/N} \tag{1.6}$$

This image is separable and it is composed of two factors: an amplitude, $A[m]$, that varies only along the vertical dimension, and a wave with a frequency, $f[n]$, that varies along the horizontal component. We set the amplitude so that it is A_{max} in the lowest part of the image and it decreases logarithmically to A_{min} at the top. And the frequency function is defined as an increasing function that starts from 0 and grows up to 60 (with $N = 2048$ being the number of horizontal pixels in the image). This image is shown in figure 1.1 and it is also called the Campbell & Robson chart. It shows a signal with a wave that oscillates slow at the left and faster towards the right and that has high contrast a the bottom and loses contrast towards the top becoming invisible.

The first sign that our visual system is non-linear is that we do not perceive the amplitude as changing logarithmically from top to bottom. It feels more linear. This is because our photo-receptors compute the log of the incoming intensity (approximately).

What is interesting is that figure 1.1 is not perceived as being separable. If you trace the region where the sine wave seems to disappears you will trace a curve. In fact, your visual system is behaving as a band-pass filter: you are sensitive to middle spatial frequencies (with a pick around 6 cycles/degree) and you are less sensitive for very low spatial frequencies (left of the image) and to high-spatial frequencies (right of the image). This curve is called the contrast sensitivity function (CSF) in the psychophysics literature and is closely related to the transfer function of the filter.

The CSF is not a simple linear function but it can be approximated by one under certain conditions. However, the CSF changes depending of many factors such as overall intensity (the pick moves towards the left under low illumination.), adaptation (long exposure to one frequency reduces the sensitivity for that frequency), age, ...

The DFT and use of sine waves became very popular in the study of visual psychophysics.

1.3   Spatial convolutional filters

We will start by studying several important linear spatially invariant filters. These filters can be implemented as convolutions between an input image and a linear kernel.

1.3.1   Box filter

Let's start with a very simple filter, the box filter. In the box filter, each output pixel is the sum of the input pixels around its neighborhood. The box convolution kernel can be written as:

$$h_{N,M}[n,m] = \begin{cases} 1 & \text{if } -N \le n \le N \text{ and } -M \le m \le M \\ 0 & \text{otherwise} \end{cases} \tag{1.7}$$

The box filter is separable as it can be written as the convolution of two 1D kernels: $h_{N,M}[n,m] = h_{N,0} \circ h_{0,M}$.

Filtering an image with the box filter results in blurring the picture. Figure 1.2 shows some box filters and the corresponding output images. Figure 1.2 (a) shows an image convolved with a uniform, rectangular kernel. Each pixel is an average of the input pixels within the rectangle. Figure 1.2 (b) and (c) show the results of blurring in just one direction. Blur happens very often in real life. It happens when we look at something very far away, or at some detail inside a picture, or when we remove our eyeglasses (or wear some that are not ours).

The box filter is simple but it has a number of issues:

- If you convolve two boxes you get a triangle. You can easily check this. For instance, in the simple case where $N = 1, M = 0$:

$$h_{1,0} \circ h_{1,0} = [1,1,1] \circ [1,1,1] = [1,2,3,2,1] \tag{1.8}$$

  the output is a triangular filter with length $2 \times L - 1$, with $L = 3$ the length of the box filter. Although that is not a problem at first sight, it means that if you blur an image twice with box filters, what you get is not equivalent to blurring only once with a larger box filter.

- The box filter is not a perfect blurring filter. A blur filter should attenuate high spatial frequencies with stronger attenuation for higher spatial frequencies. However, if you consider the highest spatial frequency, which will be an oscillating signal that takes successively on the values 1 and -1: $[..., 1, -1, 1, -1, 1, -1, ...]$ when filtered with the box filter $h_{1,0}$ the results is the same signal! However, if you filter a wave with lower frequency such as $[..., 0.5, 0.5, -1, 0.5, 0.5, -1, ...]$ then the result is $[..., 0, 0, 0, 0, ...]$. Therefore, the attenuation is not monotonic with spatial frequency as it is shown in figure 1.3. This is not a desirable behavior for a blurring filter and it can cause artifacts to appear. This could be addressed using an even size box filter $[1, 1]$. However, an even box filter is not centered around the origin and the output image will have a half-pixel translation. Therefore, odd filter sizes are preferred.

The next blurring filter addresses both of these issues.
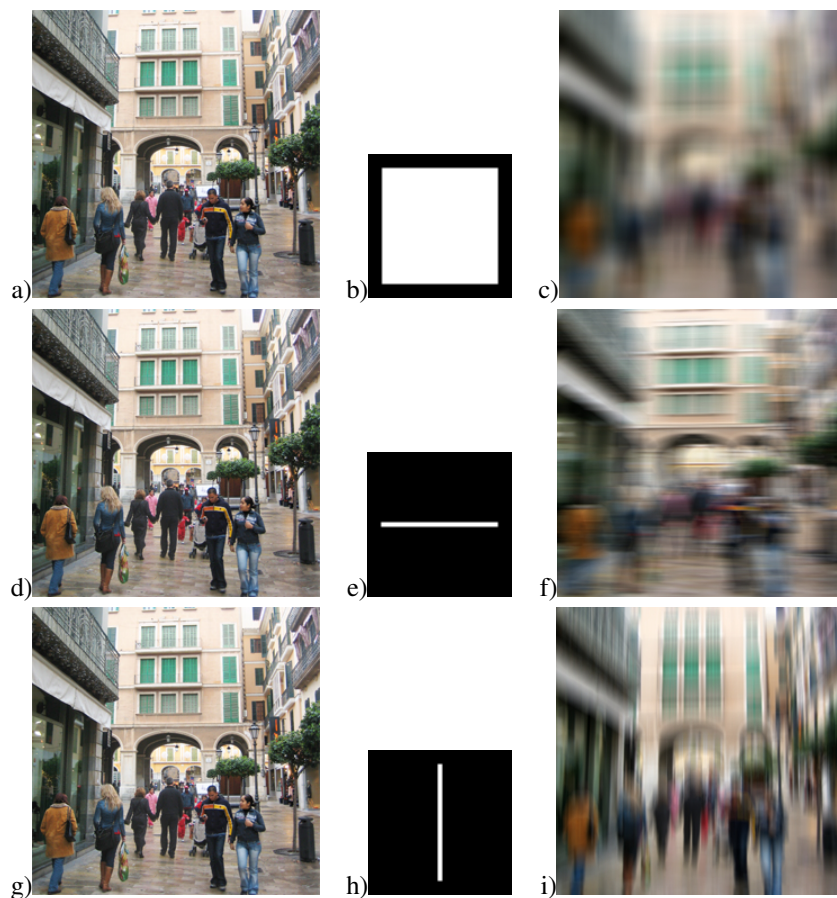
Figure 1.2
Blurring with (a) a rectangle, and a (b) horizontal and (c) vertical line. Note the structures that are both averaged out (along the direction of blurring), and maintained (perpendicular to that direction) in each resulting image.
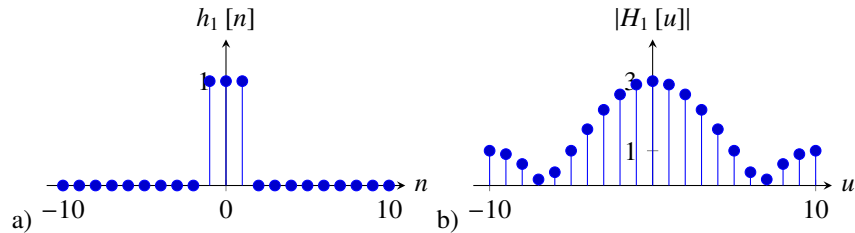
Figure 1.3

a) A one dimensional box filter ([1, 1, 1]) and b) its Fourier transform over 20 samples. Note that the frequency gain is not monotonically decreasing with spatial frequency.

1.3.2   Gaussian filter

One of the important blurring (low-pass) filters in computer vision is the gaussian filter. The gaussian filter is important because it is a good model for many naturally occurring filters. It also has a number of properties, as we will discuss here, that make it unique.

The gaussian distribution is defined in continuous variables. In one dimension:

$$g(x; \sigma) = \frac{1}{\sqrt{2\pi\sigma^2}} \exp{-\frac{x^2}{2\sigma^2}} \tag{1.9}$$

and in 2 dimensions:

$$g(x, y; \sigma) = \frac{1}{2\pi\sigma^2} \exp{-\frac{x^2 + y^2}{2\sigma^2}} \tag{1.10}$$

The parameter σ adjusts the spatial extend of the gaussian. The normalization constant is set so that the function integrates to 1.

In order to use this filter in practice we need to consider discrete locations and also approximate the function by a finite support function. In practice, we only need to consider samples within three standard deviations $x \in (-3\sigma, 3\sigma)$. At 3σ the amplitude of the gaussian is around 1% of its central value. Unfortunately, many of the properties of the gaussian that we will discuss later are only true in the continuous domain and are only approximated when using its discrete form.

For a given standard deviation parameter, σ, the discretized gaussian kernel is $g[m, n; \sigma]$:

$$g[m, n; \sigma] = \exp{-\frac{m^2 + n^2}{2\sigma^2}} \tag{1.11}$$

We have removed the normalization constant as the sum of the discrete gaussian will be different from the integral of the continuous function. So here we prefer to use the form in which the value at the origin is 1. In practice, we should normalize the discrete gaussian by the sum of its values to make sure that the gain at DC is 1.
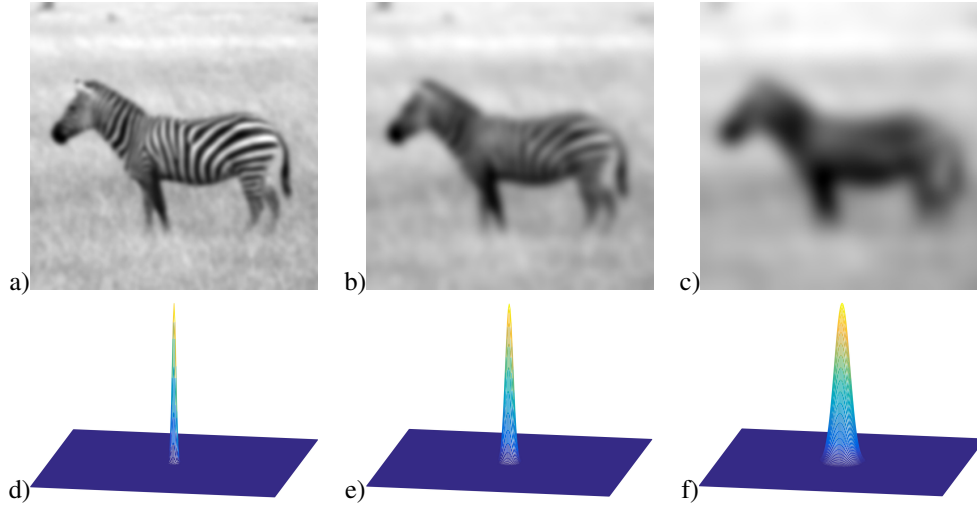
Figure 1.4
An image filtered with three gaussians with standard deviations: a) $\sigma = 2$, b) $\sigma = 4$, and c) $\sigma = 8$. Plots (d) - (f) show the three Gaussians over the same spatial support as the image. The discrete Gaussians are approximated by sampling the continuous Gaussian. The convolutions are perform with mirror boundary conditions.

By adjusting the standard deviation, σ, of the Gaussian, it is possible to adjust the level of image detail that appears in the blurred image. Figure 1.4 shows the result of narrow and wider Gaussians applied to an image.

The multi-dimensional Gaussian filter has the additional computational advantage that it can be applied as a concatenation of 1-d Gaussian filters. This can be seen by writing the 2-d Gaussian, Eq. (1.11), in the convolution equation, Eq. (**??**). Letting g^x and g^y be the 1-d Gaussian convolution kernels in the horizontal and vertical directions, we have

$$
\begin{aligned}
g[m,n] \circ f[m,n] &= \sum_{k,l} g[m-k, n-l] f[k,l] \\
&= \sum_{k,l} \exp -\frac{(m-k)^2 + (n-l)^2}{2\sigma^2} \circ f[m,n] \\
&= \sum_{k} \exp -\frac{(m-k)^2}{2\sigma^2} \left(\sum_{l} \exp -\frac{(n-l)^2}{2\sigma^2} f[k,l] \right) \\
&= g^x \circ (g^y \circ f[m,n])
\end{aligned}
$$

This can save quite a bit in computation time when applying the convolution of Eq. (1.12). If the 2-d convolution kernel is *nxn* samples, then a direct convolution of that 2-d kernel scales in proportion to n^2, since Eq. (1.12) requires one multiplication per image position

per kernel sample. Using the cascade of two 1-d kernels, resulting in an equivalent 2-d filter of the same size, scales in proportion to $2n$.

Another application of blurring is to remove distracting high-resolution image details. Fig. 1.7 shows a Gaussian low-pass filter applied to remove unwanted image details (the blocky artifacts) from an image.

Properties of the Gaussian filter:

- The n-dimensional Gaussian is the only completely circularly symmetric operator that is separable.
- The continuous Fourier transform of a gaussian is also a gaussian.

$$G(u, v; \sigma) = \exp -2\pi^2 (u^2 + v^2)\sigma^2 \qquad (1.12)$$

Note that this function is monotonically decreasing in magnitude for increasing frequencies, and it is also radially symmetric.

- The convolution of two n-dimensional gaussians is an n-dimensional gaussian.

$$g(x, y; \sigma_1) \circ g(x, y; \sigma_2) = g(x, y; \sigma_3) \qquad (1.13)$$

where the variance of the result is the sum $\sigma_3^2 = \sigma_1^2 + \sigma_2^2$. This is a remarkable property of gaussian filters and is the basis of the gaussian pyramid that we will see later. To prove this property, one can use the Fourier transform of the gaussian and the fact that the convolution is the product of Fourier transforms.

- The gaussian is the solution to the heat equation.
- Repeated convolutions of any function concentrated in the origin result in a gaussian (central limit theorem).
- In the limit $\sigma \to 0$ the Gaussian becomes an impulse. This property is shared by many other functions, but it is a useful thing to know.

However, many of these properties do not work for the discrete approximation $g[n, m; \sigma]$ obtained by directly sampling the values of the gaussian at discrete locations. To see this let's look at some examples. Let's consider a gaussian with variance $\sigma^2 = 1/2$. It can be approximated by 5 samples. We will call this approximation g_5:

$$g_5[n] = [0.0183, 0.3679, 1.0000, 0.3679, 0.0183] \qquad (1.14)$$

first, note when convolved with a wave $[1, -1, 1, -1, ...]$ the result is not zero. This is to be expected from the form of the FT of the gaussian. You can check that if you compute the approximation for $\sigma^2 = 1$ by discretizing the gaussian, the result obtained is not equal to doing $g_5 \circ g_5$. Therefore, as you apply successive convolutions of gaussian the errors will accumulate.

In practice, there are very efficient approximations to the Gaussian filter for certain σ values with nicer properties than when working with discretized gaussians. One common

approximation of the gaussian filter is to use binomial coefficients **??**. Binomial coefficients provide a compact approximation of the gaussian coefficients using only integers. The binomial coefficients use the central limit theorem to approximate a gaussian as successive convolutions of a very simple function. The simplest low-pass filter is the box filter $[1, 1]$. The binomial coefficients form the Pascal's triangle as shown in figure 1.5.

b_0						1						$\sigma_0^2 = 0$
b_1					1		1					$\sigma_1^2 = 1/4$
b_2				1		2		1				$\sigma_2^2 = 1/2$
b_3			1		3		3		1			$\sigma_3^2 = 3/4$
b_4		1		4		6		4		1		$\sigma_4^2 = 1$
b_5	1		5		10		10		5		1	$\sigma_5^2 = 5/4$
b_6	1	6		15		20		15		6	1	$\sigma_6^2 = 3/2$
b_7	1	7	21		35		35		21	7	1	$\sigma_7^2 = 7/4$
b_8	1	8	28	56		70		56	28	8	1	$\sigma_8^2 = 2$

Figure 1.5

Binomial coefficients. To build the Pascal's triangle, each number is the sum of the number above to the left and the one above to the right.

The sum of all the coefficients for each binomial filter b_n is 2^n, and their variance is $\sigma^2 = n/4$. One remarkable property of the binomial filters is that $b_n \circ b_m = b_{n+m}$, and, therefore, $\sigma_n^2 + \sigma_m^2 = \sigma_{n+m}^2$, which is the analogous to the gaussian property in the continuous domain. Note that the values of b_2 are different from g_5 despite that both will be used as approximations to the same gaussian.

The simplest approximation to the Gaussian filter is the 3-tap kernel:

$$b_2 = [1, 2, 1] \tag{1.15}$$

This filter is interesting because it is even (so it can be applied to an image without producing any translation) and its DFT is:

$$B_2[u] = 2 + 2\cos(2\pi u/N) \tag{1.16}$$

it has a monotonic amplitude gain in frequency (there are no ripples) as shown in figure 1.6. All the even binomial filters can be written as successive convolutions with the kernel $[1, 2, 1]$. Therefore, their Fourier transform is a power of the Fourier transform of the filter $[1, 2, 1]$ and therefore they are also monotonic:

$$B_{2n}[u] = (2 + 2\cos(2\pi u/N))^n \tag{1.17}$$

For all the binomial filters b_n, when they are convolved with the wave $[1, -1, 1, -1, ...]$, the result is the zero signal $[0, 0, 0, 0, ...]$. This is a very nice property of binomial filters and will become very useful later when talking about downsampling an image (see section **??**).
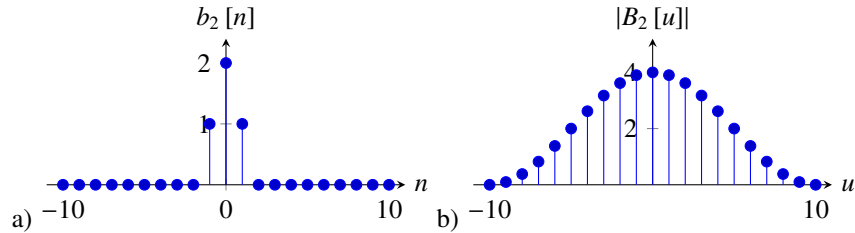
Figure 1.6
a) A one dimensional 3-tap approximation to the gaussian filter ($[1, 2, 1]$) and b) its Fourier transform over 20 samples. Note that the frequency gain is decreasing monotonically with spatial frequency and it becomes zero at the highest frequency, $G_3[10] = 0$.

The gaussian in 2D can be approximated, using separability, as the convolution of two binomial filters one vertical and another horizontal. For instance:

$$b_{2,2} = b_{2,0} \circ b_{0,2} = \begin{bmatrix} 1 & 2 & 1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.18}$$

What is the inverse? can we remove gaussian filter? Show the matrix for some simple case and show it does not behave nicely.

Due to all these properties the gaussian and the binomial filter are extensively used in computer vision.

1.3.3   Image derivatives

Computing image derivatives is an essential operator for extracting useful information from images. As we show in the previous chapter, image derivatives allowed us computing boundaries between objects and to have access to some of the 3D information lost when projecting the 3D world into the camera plane. Derivatives are useful because they give us information about where are happening the changes in the image and we expect those changes to be correlated with transitions between objects.

If we had access to the continuous image, then image derivatives could be computed as: $\partial \mathbf{I}(x, y) / \partial x$. However, there are two reasons why we might not be able to apply this definition

• the first one is that we only have access to a sampled version of the input image: $\mathbf{I}[m, n]$
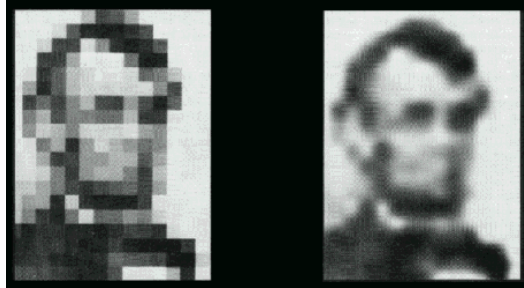
Figure 1.7

Left: input image. Right: blurred version. The left version has many spurious details introduced by the undersampling of the image. The right image has been blurred by a large Gaussian filter. ((image from http://acor.org/sgreene/hmsbeagle/html/content/17/recroom/artgalas.htm, after 1973 image by Bela Julesz and Leon Harmon)).

· even if we had access to the continuous image, the image could contain many non-derivable points and the gradient would not be defined. We will see how to address this issue later when we study gaussian derivatives.

for now, let's focus on the problem of approximating the continuous derivative with discrete operators. As the derivative is a linear operator, it can be approximated by a discrete linear filter. There are several ways in which image derivatives can be approximated.

Let's start with a simple approximation to the derivative operator that we have already played with: $d_0 = [1, -1]$. In one dimension, convolving a signal $f[n]$ with this filter results in:

$$f \circ d_0 = f[n] - f[n-1] \tag{1.19}$$

this approximates the derivative by the difference between consecutive values. Figure 1.8.c shows the result of filtering a 1-d signal (fig.1.8.a) convolved with $d_0[n]$ (fig.1.8.b). The output is zero wherever the input signal is constant and it is large in the places where there are variations in the input values. However note that the output is not perfectly aligned with the input. In fact there is a half a sample displacement to the right. This is due to the fact that $d_0[n]$ is not centered around the origin.

This can be addressed with a different approximation to the spatial derivative: $d_1 = [1, 0, -1]/2$. In one dimension, convolving a signal $f[n]$ with $d_1[n]$ results in:

$$f \circ d_1 = \frac{f[n+1] - f[n-1]}{2} \tag{1.20}$$

Figure 1.8.e shows the result of filtering the 1-d signal (fig.1.8.a) convolved with $d_1[n]$ (fig.1.8.d). Now the output shows the highest magnitude output in the mid point where there is variation in the input signal.
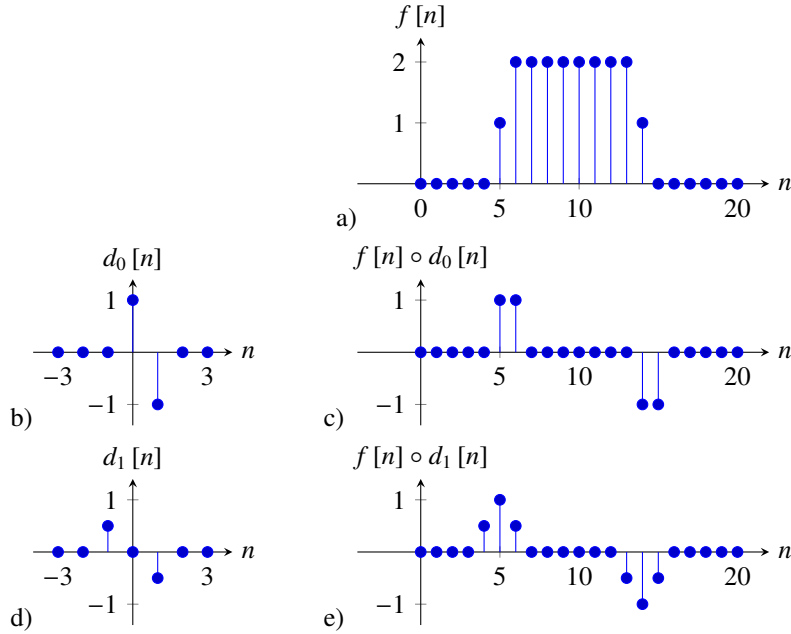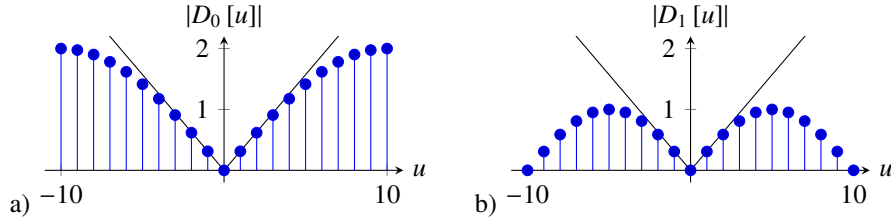
Figure 1.8
a) b).

It is also interesting to see the behavior of the derivative and its discrete approximations in the Fourier domain.

$$df(x)/dx \leftrightarrow jwF(w) \tag{1.21}$$

in the continuous Fourier domain, derivation can be done by multiplying by jw. The DFT of $d_0[n]$ is:

$$
\begin{aligned}
D_0[u] &= 1 - \exp\left(-2\pi j\frac{u}{N}\right) = \\
&= \exp\left(-\pi j\frac{u}{N}\right)\left(\exp\left(\pi j\frac{u}{N}\right) - \exp\left(-\pi j\frac{u}{N}\right)\right) \\
&= \exp\left(-\pi j\frac{u}{N}\right)2j\sin(\pi u/N)
\end{aligned}
\tag{1.22}
$$

the first term is a pure phase shift and it is responsible of the half a sample delay in the output. The second term is the amplitude gain and it can be approximated by a linear dependency on u for small u values.

Figure 1.9

Magnitude of a) $D_0[u]$ and b) $D_1[u]$ and comparison with $|2\pi u/N|$, shown as a thin black line. Both DFT are computed over 20 samples.

The DFT of $d_1[n]$ is:

$$D_1[u] = 1/2 \exp\left(2\pi j\frac{u}{N}\right) - 1/2 \exp\left(-2\pi j\frac{u}{N}\right) =$$
$$= j\sin(2\pi u/N)$$

(1.23)

Figure 1.10 shows the magnitude of $D_0[u]$ and $D_1[u]$ and compares it with $|2\pi u/N|$ which will be the ideal approximation to the derivative. The amplitude of $D_0[u]$ provides a better approximation to the ideal derivative, but the phase of $D_0[u]$ introduces a small shift in the output. On the other hand, $D_1[u]$ has no shift, but it approximates the derivative over a smaller range of frequencies. The output to $D_1[u]$ is smoother than the output to $D_0[u]$, and, in particular, $D_1[u]$ gives a zero output when the input is the signal $[1, -1, 1, -1, ...]$. In fact, we can see that $[1, 0, -1] = [1, -1] \circ [1, 1]$, and, therefore $D_1[u] = D_0[u]\,B_1[u]$, where $B_1[u]$ is the DFT of the binomial filter $b_1[n]$.

Derivates have become an important tool to represent images and they can be used to extract a great deal of information from the image as it was shown in the previous chapter. One thing about derivatives is that it might seem as we are loosing information from the input image. An important question is: if we have the derivative of a signal, can we recover the original image? what information is being lost? Intuitively, we should be able to recover the input image by integrating its derivative, but it is an interesting exercise to look in detail how this integration can be performed. We will start with a one dimensional signal and then we will discuss the two dimensional case.

A simple way of seeing that we can recover the input from its derivatives is to write the derivative in matrix form. This is the matrix that corresponds to the convolution with the kernel $[1, -1]$ that we will call D_0. The next two matrices show the matrix D_0 and its

inverse D_0^{-1} for a 1D image of length 5 pixels using zero boundary conditions:

$$
D_0 = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \qquad D^{-1} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{1.24}
$$

We can see that the inverse D^{-1} is reconstructing each pixel as a sum of all the derivate values from the left-most pixel to the right. And the inverse perfectly reconstructs the input. But, this is cheating because the first sample of the derivative gets to see the actual value of the input signal and then we can integrate back the entire signal. That matrix is assuming zero boundary conditions for the signal and the boundary gives us the needed constraint to be able to integrate back the input signal.

But what happens if you only get to see differences and you remove any pixel that was affected by the boundary? In this case, the derivative operator in matrix form is:

$$
D_0 = \begin{bmatrix} -1 & 1 & 0 & 0 & 0 \\ 0 & -1 & 1 & 0 & 0 \\ 0 & 0 & -1 & 1 & 0 \\ 0 & 0 & 0 & -1 & 1 \end{bmatrix} \tag{1.25}
$$

Let's consider the next 1D input signal:

$$
g = [1, 1, 2, 2, 0] \tag{1.26}
$$

then, the output of the derivative operator is:

$$
f = D_0 g = [0, -1, 0, 2] \tag{1.27}
$$

Note that this vector has one sample less than the input.

To recover g we can not invert D_0 as it is not a square matrix, but we can compute the pseudo inverse which turns out to be:

$$
D_0^+ = \frac{1}{5} \begin{bmatrix} -4 & -3 & -2 & -1 \\ 1 & -3 & -2 & -1 \\ 1 & 2 & -2 & -1 \\ 1 & 2 & 3 & -1 \\ 1 & 2 & 3 & 4 \end{bmatrix} \tag{1.28}
$$

It has a funny structure and it is easy to see how it can be written in the general form for signals of length N. Also note that this is not a convolution. Another important thing is that the inversion process is trying to recover more samples than there are observations. The

trade off is that the signal that it will recover will have zero mean (so it looses one degree of freedom that can not be estimated). In this example, the reconstructed input is:

$$\hat{g} = D^+ f = [-0.2, -0.2, 0.8, 0.8, -1.2] \tag{1.29}$$

Note that $\sum \hat{g} = 0$ and that this is $\hat{g} = g - 1.2$, where 1.2 is the mean value of samples on g. Then, you still can recover the input signal up to the DC component.

In two dimensions things are a bit more complex. There are several ways in which partial image derivatives can be approximated. For instace, we can compute derivatives along the n and m components.

$$\begin{bmatrix} 1 \\ -1 \end{bmatrix} \begin{bmatrix} 1 - 1 \end{bmatrix} \tag{1.30}$$

Or we can use a rotated reference frame as it is done in the Robert-Cross operator, introduced in 1963 in a time when reading an image of 256×256 pixels into memory took several minutes:

$$\begin{bmatrix} 1 & 0 \\ 0 & -1 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ -1 & 0 \end{bmatrix} \tag{1.31}$$

Although these operators are very old and better algorithms exist nowadays for edge extraction, when an efficient solution is needed, these simple operators are still very useful and had been used as key operators in modern computer vision descriptors such as HOG as we will see later.

Show some applications doing edge edition. This is a non-linear filter.

FIGURE: a) removing one object, b) moving one object from one image to another.

1.3.4   Gaussian derivatives

In the previous section we studied how to discretize derivatives. However, computing derivatives in practice has several difficulties. First, derivatives are sensitive to noise. In the presence of noise, as images tend to vary slowly, the difference between two continuous pixel values will be dominated by noise. There are also situations in which the derivative of an image is not defined. For instance, consider an image in the continuous domain with the form: $\mathbf{I}(x, y) = 0$ if $x < 0$ and 1 otherwise. If we try to compute $\partial \mathbf{I}(x, y)/\partial x$ we will get 0 everywhere, but around $x = 0$ the value of the derivative is not defined. We avoided this issue in the previous section as for discrete images the approximation of the derivative is always defined.

Gaussian derivatives address these two issues. They where introduced by Koendering and Van Doorm as a model of the processing perform by neurons in the visual system.
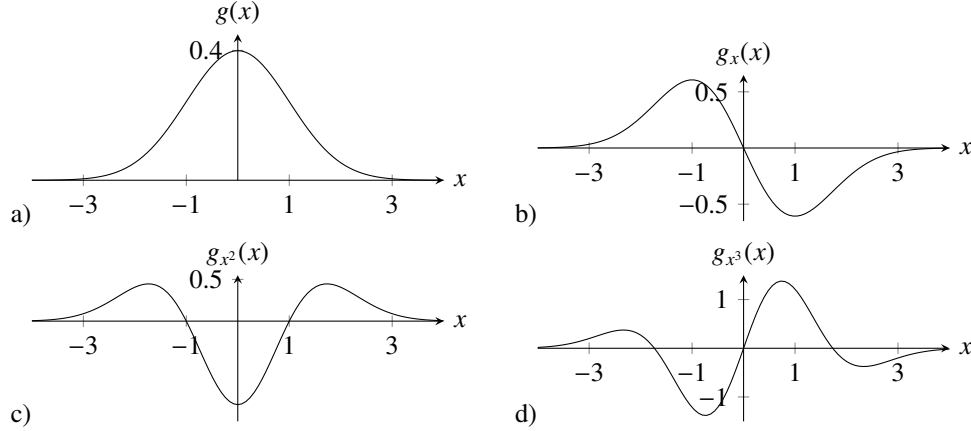
Figure 1.10
a) 1D Gaussian with $\sigma = 1$ and b-d) its derivatives up to order 3.

Let's start with the following observation: For two functions defined in the continuous domain $f(x, y)$ and $g(x, y)$, we can write:

$$\frac{\partial f(x, y)}{\partial x} \circ g(x, y) = f(x, y) \circ \frac{\partial g(x, y)}{\partial x} \tag{1.32}$$

this is easy to prove in the Fourier domain. If our goal is to compute image derivatives and then blur the output using a low-pass filter, $g(x, y)$, then, instead of computing the derivative of the image we can compute the derivatives of the filter kernel and convolve it with the image. Even if the derivative of f is not defined in some locations, we can always compute the result of this convolution.

If $g(x, y)$ is a blurring kernel it will smooth the derivatives reducing the output noise at the expense of a loss in spatial resolution. If g is a Gaussian, then the derivative is:

$$g_x(x, y; \sigma) = \frac{\partial g(x, y; \sigma)}{\partial x} = \frac{-x}{2\pi\sigma^4} \exp{-\frac{x^2 + y^2}{2\sigma^2}} = \frac{-x}{\sigma^2} g(x, y; \sigma) \tag{1.33}$$

and the second order derivative is:

$$g_{x^2}(x, y; \sigma) = \frac{x^2 - \sigma^2}{\sigma^4} g(x, y; \sigma) \tag{1.34}$$

Each derivative can be written as the product between a polynomial on x, with the same order as the derivative, times a gaussian. The family of polynomials that result on computing gaussian derivatives is called Hermite polynomials. The general expression for the n
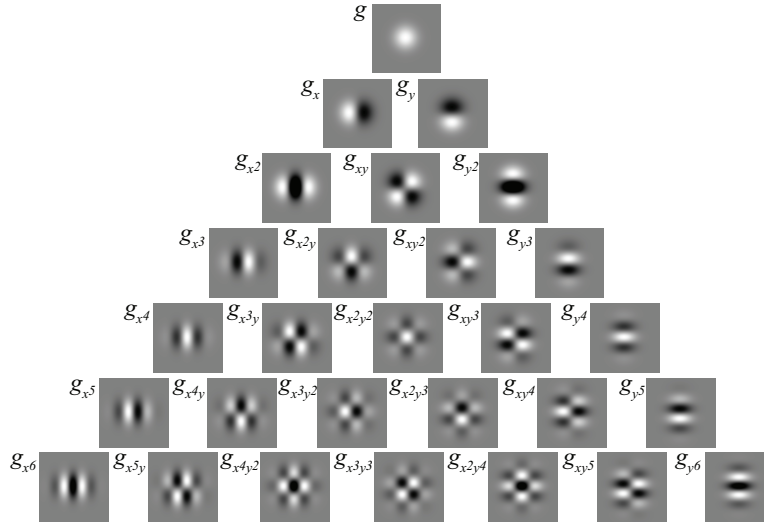
Figure 1.11
Gaussian derivatives up to order 6. All the kernels are separable. Although the seem similar to Fourier basis fig. 1.12 shows that they are different to sine and cosine waves, instead they look more like products of cosine and sine waves.

derivative of a gaussian is:

$$g_{x^n}(x; \sigma) = \frac{\partial^n g(x)}{\partial x^n} = \left(\frac{-1}{\sigma \sqrt{2}}\right)^n H_n\left(\frac{x}{\sigma \sqrt{2}}\right) g(x; \sigma) \tag{1.35}$$

the first Hermite polynomial is $H_0(x) = 1$, for $n = 0$ we have the original Gaussian. Figure 1.10 shows the 1D Gaussian derivatives.

In two dimensions, as the gaussian is separable, the partial derivatives result on the product of two Hermite polynomial, one for each dimension:

$$g_{x^n, y^m}(x, y; \sigma) = \frac{\partial^{n+m} g(x, y)}{\partial x^n \partial y^m} = \left(\frac{-1}{\sigma \sqrt{2}}\right)^{n+m} H_n\left(\frac{x}{\sigma \sqrt{2}}\right) H_m\left(\frac{y}{\sigma \sqrt{2}}\right) g(x, y; \sigma) \tag{1.36}$$
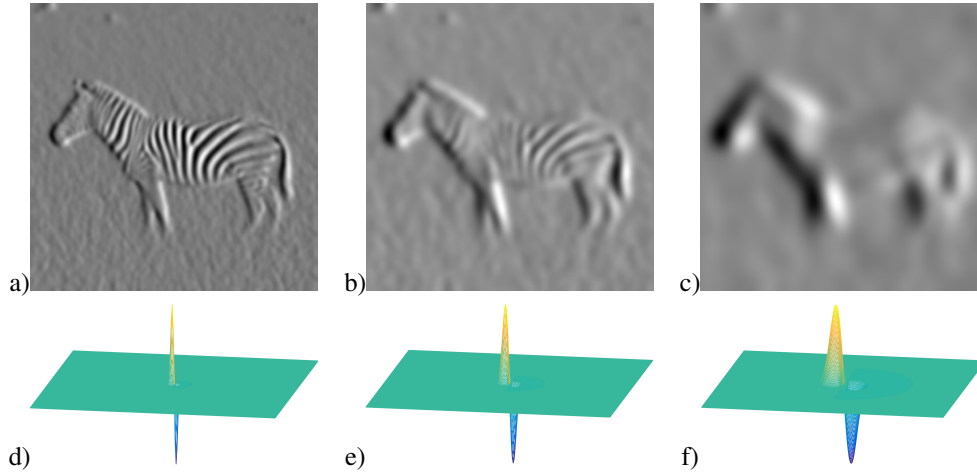
Figure 1.11 shows the 2D Gaussian derivatives, and Figure 1.12 shows the corresponding Fourier transforms.

The Gaussian derivatives share many of the properties of the Gaussian:

- The convolution of two Gaussian derivatives of order n and m and variances σ_1^2 and σ_2^2 results in another gaussian derivative of order $n + m$ and variance $\sigma_1^2 + \sigma_2^2$. Proving this property on the spatial domain can be tedious. However it is trivial to prove it in the Fourier domain.
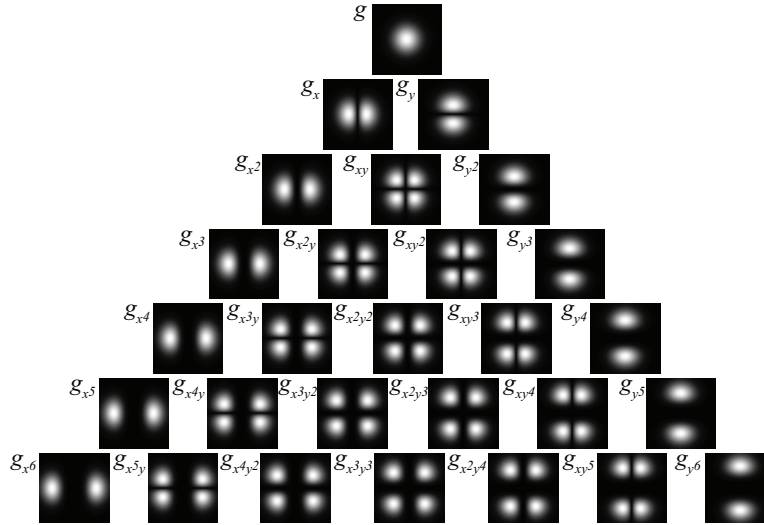
Figure 1.12

Fourier transform of the Gaussian derivatives shown in figure 1.11. Units are arbitrary.

Figure 1.13

An image filtered with three Gaussian derivatives with standard deviations: a) $\sigma = 2$, b) $\sigma = 4$, and c) $\sigma = 8$. In the filtered images, bright pixels correspond to positive values and dark pixels correspond to negative values. Plots (d) - (f) show the three Gaussian derivatives over the same spatial support as the image. The discrete functions are approximated by sampling the continuous Gaussian derivatives. The convolutions are perform with mirror boundary conditions.

Figure 1.13 shows an image filtered with three gaussian derivatives with different widths. Interestingly, derivatives at different scales emphasize different aspects of the image. The fine-scale derivatives (fig. 1.13.a) highlight the bands in the zebra, while the coarse-scale derivatives (fig. 1.13.c) emphasize more the object boundaries. This multiscale image analysis will be studied in depth in the following chapter.

When processing images we have to use discrete approximations for the Gaussian derivatives. After discretization, many of the properties of the continuous Gaussian will not hold exactly.

There are many discrete approximations. For instance, we can take samples of the continuous functions. In practice it is common to use the discrete approximation given by the binomial filters. Figure 1.14 shows the result of convolving the binomial coefficients with: $b_n \circ [1, -1]$.

d_0				1		−1			
d_1			1		0		−1		
d_2		1		1		−1		−1	
d_3		1	2		0		−2	−1	
d_4	1	3		2		−2		−3	−1
d_5	1	4	5		0		−5	−4	−1

Figure 1.14
Derivative of binomial coefficients resulting from convolving $b_n \circ [1, -1]$. The first two filters are the ones we have studied in detail in the previous section.

In two dimensions, we can use separable filters and build a partial derivative as:

$$Sobel_x = \begin{bmatrix} 1 & 0 & -1 \end{bmatrix} \circ \begin{bmatrix} 1 \\ 2 \\ 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & -1 \\ 2 & 0 & -2 \\ 1 & 0 & -1 \end{bmatrix} \tag{1.37}$$

$$Sobel_y = \begin{bmatrix} -1 & -2 & -1 \\ 0 & 0 & 0 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.38}$$

This particular filter is called the Sobel-Feldman operator. The goal of this operator was to be compact and to be as isotropic as possible. The sobel-feldman operator can be implemented very efficiently as it can be written as the convolution with 4 small kernels: $Sobel_x = b_1 \circ d_0 \circ b_1^T \circ b_1^T$. The DFT is:

$$Sobel_x [u, v] = D_1 [u] B_2 [v] = j \sin(2\pi u/N)(2 + 2\cos(2\pi v/N)) \tag{1.39}$$
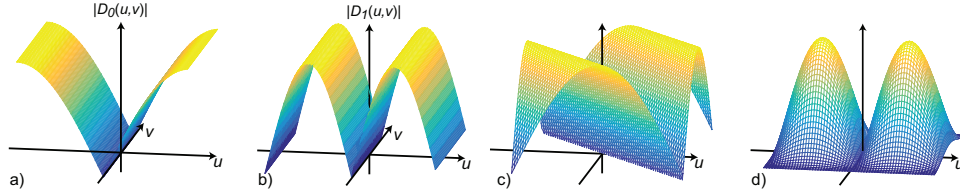
Figure 1.15
Magnitude of the DFT of four different discretization of Gaussian derivatives: a) d_0, b) d_1, c) Cross-Robert operator and d) Sobel-Feldman operator. As d_0 and d_1 are one dimensional their 2D-DFT vary only along one dimension. The Cross-Robert operator is similar to a rotated version of d_1. The Sobel-Feldman operator has the profile of D_1 along the axis $u = 0$ and it is proportional to the profile of B_2 along any section $v = constant$.

and it is shown in fig. 1.15.d. $N \times N$ is the extension of the domain (the operator is zero padded).

Fig. 1.15 compares the DFT of the 4 types of approximations of the derivatives that we have discussed. These operators are still very popular. *S obel* has the best tolerance to noise due to its band-pass nature. The kernel d_0 is the one that provides the highest resolution in the output. Fig. 1.16 shows the output of different derivative approximations to a simple input image containing a circle. In the next section we will discuss now to use these derivatives to extract other interesting quantities.

1.3.5   Image gradient and directional derivatives

As we saw in chapter one, an important image representation is given by the image gradient. From the image derivatives we can define also the image gradient as the vector:

$$\nabla \mathbf{I} = \left(\frac{\partial \mathbf{I}}{\partial x}, \frac{\partial \mathbf{I}}{\partial y} \right) \tag{1.40}$$

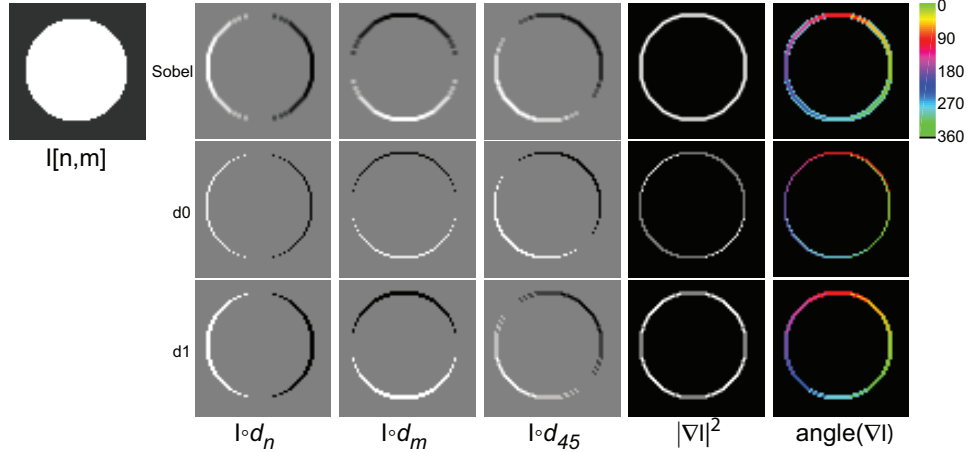For each pixel, the output is a two dimensional vector. In the case of using gaussian derivatives, we can write:

$$\nabla \mathbf{I} \circ g = \left(g_x(x, y), g_y(x, y) \right) \circ \mathbf{I} = \nabla g \circ \mathbf{I} \tag{1.41}$$

Although we have mostly computed derivatives along the *x* and *y* variables, we can obtain the derivative on any orientation as a linear combination of the two derivatives along the main axes. With $\mathbf{t} = (\cos(\theta), \sin(\theta))$, we can write the directional derivative a long the vector $\mathbf{t}$ as:

$$\frac{\partial \mathbf{I}}{\partial \mathbf{t}} = \nabla \mathbf{I} \cdot \mathbf{t} = \cos(\theta) \frac{\partial \mathbf{I}}{\partial x} + \sin(\theta) \frac{\partial \mathbf{I}}{\partial y} \tag{1.42}$$

In the gaussian case:

$$\frac{\partial \mathbf{I}}{\partial \mathbf{t}} \circ g = \left(\cos(\theta) g_x(x, y) + \sin(\theta) g_y(x, y) \right) \circ \mathbf{I} = (\nabla g \cdot \mathbf{t}) \circ \mathbf{I} = g_\theta(x, y) \circ \mathbf{I} \tag{1.43}$$

Figure 1.16
Image containing a circle filtered along the directions *n*, *m* and 45 degrees. And also the magnitude and the angle of the gradient. The angle is shown only where the magnitude is > 0. The derivative output along 45 degrees is obtained as a linear combination of the derivatives outputs along *n* and *m*. Check the differences among the different kernels. The Sobel operator gives the most rotationally invariant gradient magnitude at the expense of a bit of blurring of the output.

with $g_\theta(x, y) = \cos(\theta)g_x(x, y) + \sin(\theta)g_y(x, y)$. However, to compute the derivate a long any arbitrary angle θ does not require doing new convolutions. Instead, we can compute any derivative as a linear combination of the output of convolving the image with $g_x(x, y)$ and $g_y(x, y)$:

$$\frac{\partial \mathbf{I}}{\partial \mathbf{t}} \circ g = \cos(\theta)g_x(x, y) \circ \mathbf{I} + \sin(\theta)g_y(x, y) \circ \mathbf{I} \tag{1.44}$$

When using discrete convolutional kernels $d_n[n, m]$ and $d_m[n, m]$ to approximate the derivatives along *n* and *m*, it can be written as:

$$\nabla \mathbf{I} = (d_n[n, m], d_m[n, m]) \circ \mathbf{I} \tag{1.45}$$

and

$$\nabla \mathbf{I} \cdot \mathbf{t} = d_\theta[n, m] \circ \mathbf{I} \tag{1.46}$$

with $d_\theta[n, m] = \cos(\theta)d_n[n, m] + \sin(\theta)d_m[n, m]$. We expect that the linear combination of these two kernels should approximate the derivative in the direction θ. The quality of this approximation will vary for the different kernels we have seen in the previous sections.

- The structure tensor: this is an example of a powerful image derivative representation. And we will see more of it (or related) later.
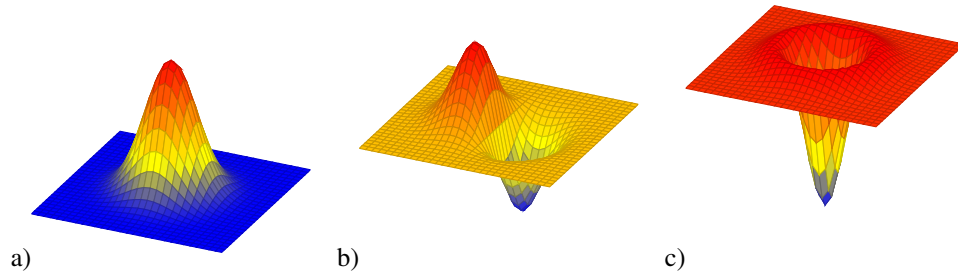
Figure 1.17
Three 2D gaussian-related convolutional kernels: a) 2D gaussian with $\sigma = 1$, b) its x derivative, and c) its laplacian.

1.3.6   Laplacian

The Laplacian filter was made popular by Marr and Hildreth in 1980 in the search for operators that locate the boundaries between objects. The Laplacian is a common operator from differential geometry to measure the divergence of the gradient and it appears frequently in modeling fields in physics. One cool example is the paper "can one hear the shape of a drum" where the laplacian is used for modeling vibrations in a drum and the sounds it produces as a function of its shape, those results have applications also in spectral methods for image segmentation as we will see later.

The Laplacian operator is defined as the sum of the second order partial derivatives of a function:

$$\nabla^2 \mathbf{I} = \frac{\partial^2 \mathbf{I}}{\partial x^2} + \frac{\partial^2 \mathbf{I}}{\partial y^2} \tag{1.47}$$

The laplacian is more sensitive to noise than the first order derivatives. Therefore, in the presence of noise, it is useful to filter the output with a gaussian. We can write:

$$\nabla^2 \mathbf{I} \circ g = \nabla^2 g \circ \mathbf{I} \tag{1.48}$$

Where

$$\nabla^2 g = \frac{x^2 + y^2 - 2\sigma^2}{\sigma^4} g(x, y) \tag{1.49}$$

Figure 1.17 compares the 2D gaussian laplacian (fig. 1.17.c) with the first order derivative (fig. 1.17.b) of a 2D gaussian (fig. 1.17.a). Due to its shape, the laplacian is also called the inverted mexican hat wavelet. Despite that visually it might seem as if the laplacian has a negative mean the mean is actually zero as the positive side is wider than the negative. Figure 1.18 shows the DFT of the gaussian laplacian for three different values of σ. For values $\sigma > 1$ the resulting filter is band-pass.
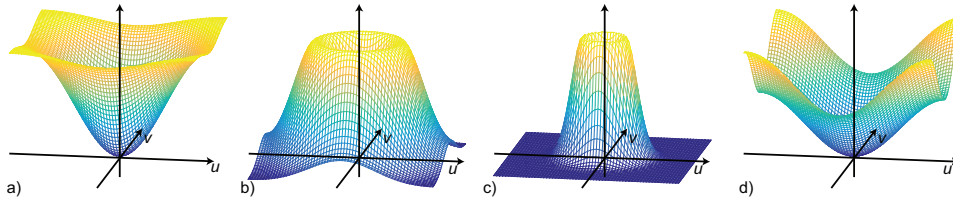
Figure 1.18
Magnitude of the DFT of the Gaussian Laplacian with three different widths and a discrete approximation: a) $\sigma = 1/2$, b) $\sigma = 1$, c) $\sigma = 2$ and d) DFT of the five-point discrete approximation to the laplacian.

In the discrete domain there are several approximations to the laplacian filter. Lindeberg explored different approximations to the image Laplacian in order to optimize its properties. In one dimension, the Laplacian can be approximated by $[1, -2, 1]$ which is the result of the convolution of $[1, -1] \circ [1, -1]$. In two dimensions, the most popular approximation is the five-point formula which consists in convolving the image with the kernel:

$$\nabla_5^2 = \begin{bmatrix} 0 & 1 & 0 \\ 1 & -4 & 1 \\ 0 & 1 & 0 \end{bmatrix} \tag{1.50}$$

It involves the central pixel and its four nearest neighbors. This is a sum-separable kernel: it corresponds to approximating the second order derivative for each coordinate and summing the result (i.e., convolve the image with $[1, -2, 1]$ and also with its transpose and summing the two outputs. It is important to shift by one pixel, on the appropriate dimension, the two results to make sure that the sum results in the Laplacian.) Figure 1.17.d shows the DFT of this approximation. It also has a quadratic from near the origin of the frequency domain, but it has a high-pass shape similar to the one obtained for a small value of σ.

Note that when summing two first order derivatives along x and y, what we obtain is a first order derivative along the 45 degrees angle. However, when summing up the two second order derivatives we obtain a rotationally invariant kernel. This is shown in fig. 1.19

The discrete approximation also provides a better intuition of how it works than its continuous counterpart. The Laplacian filter has a number of advantages with respect to the gradient:

- It is rotationally invariant: it is a linear operator that responds equally to edges in any orientation (this is only approximate in the discrete case).
- It measures curvature: if the image contains a linear trend the derivative will be non-zero despite being no boundaries, while the laplacian will be zero.
- Edges can be located as the zero-crossings in the Laplacian output. However, this way of detecting edges is not very reliable.
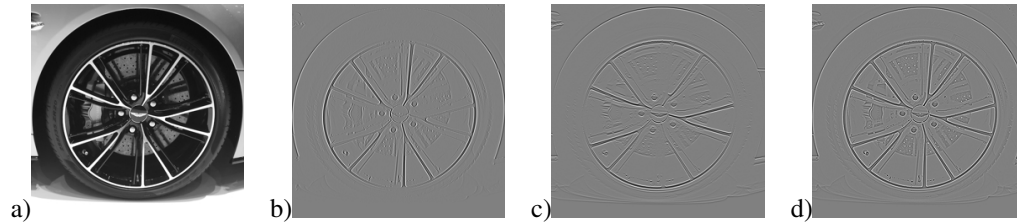
Figure 1.19

a) Input image, b) second order derivative along *x*, c) second order derivative along *y* and d) the sum of the two second order derivatives, which results in the laplacian output.

· Zero crossings of an image form closed contours.

Marr-Hildreth used zero-crossings of the laplacian output to compute edges, but this method is not used nowadays for edge detection. Instead the laplacian filter is widely used in a number of other image representations. It is used to build image pyramids (multiscale representations, chapter **??**), to detect points of interest in images (it is the basic operator used to detect keypoints where to compute SIFT descriptors, which we will discuss in chapter **??**).

Figure 1.20 compares the output of the first order derivative (Fig. 1.20.b) and the laplacian (Fig. 1.20.d) on a simple 1D signal (Fig. 1.20.a). The local maximum of the derivative output (Fig. 1.20.c) and the zero crossings of the laplacian output (Fig. 1.20.e) are aligned with the transitions (boundaries) of the input signal (Fig. 1.20.a).

The Laplacian filter can also be used as a coarse approximation of the behavior of the early visual system. When looking at the magnitude of the DFT of the laplacian with $\sigma = 1$ (fig. 1.18), the shape seems reminiscent of our subjective evaluation of our own visual sensitivity to spatial frequencies when we look at the Campbell & Robson chart (fig. 1.1). As the visual filter does not seem to cancel exactly the very low spatial frequencies as the laplacian does, a better approximation is:

$$h = \nabla^2 g + \lambda g \qquad (1.51)$$

where *h* is the approximate impulse response of the human visual system, λ is a small constant that is equal to the DC gain of the filter. This particular form of the human sensitivity function helps to explain some visual illusions. One of them is shown in figure 1.21. This visual illusion is called the Vasarely visual illusion. Images (a) and (d) show two gray-scale pyramids formed by superimposing squares of increasing (or decreasing) intensity. When looking at them we perceive the diagonal directions as being brighter (a) or darker (d) than their neighborhood. This is an illusion because there is not such a difference in image intensity. One explanation consists in saying that the image that we perceive is the output of a filter (as shown in eq. 1.51). Images (b) and (e) show the output of such a
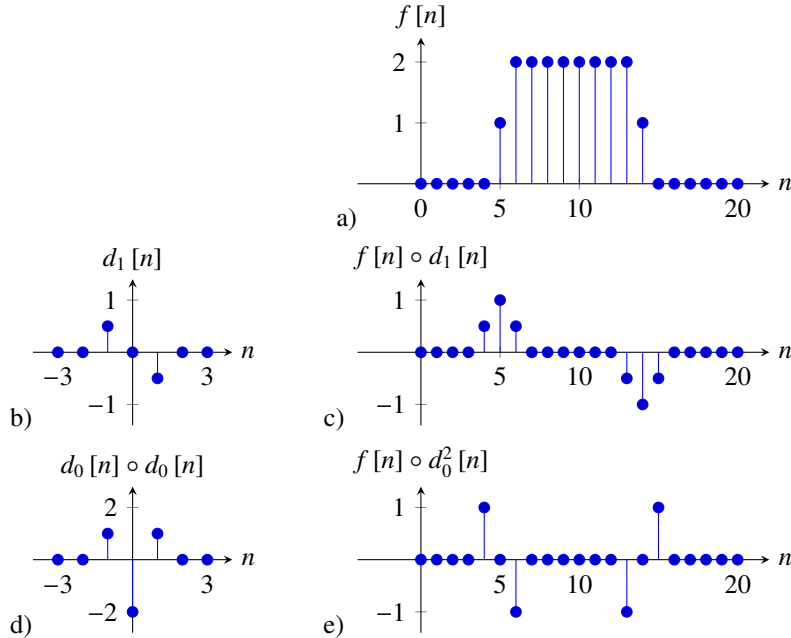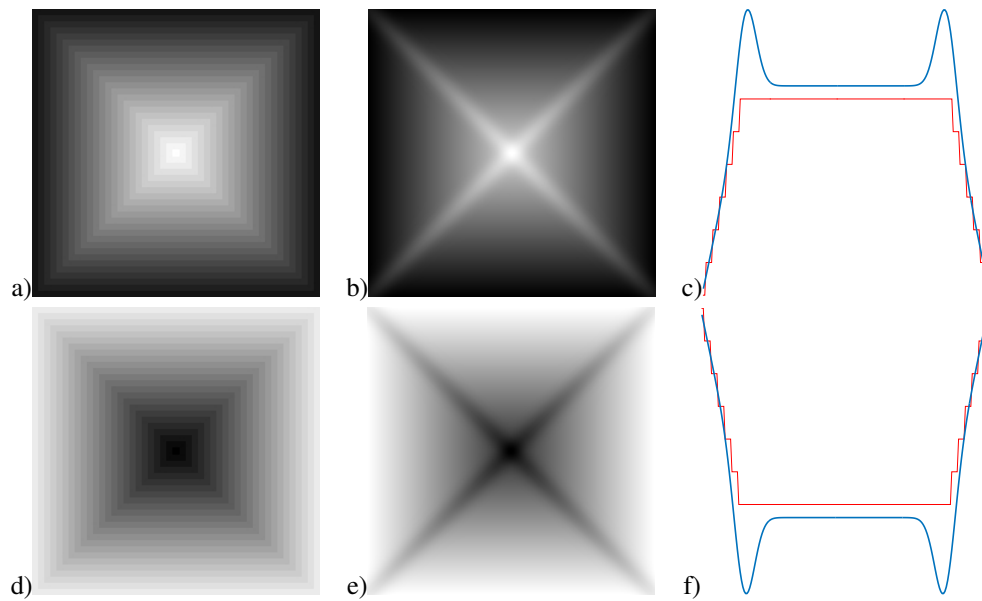
Figure 1.20

Comparison between the output of a first order derivative and the Laplacian of 1D signal. a) Input signal, b) kernel d_1, c) output of the derivative (convolution of (a) and (b)), d) discrete approximation of the Laplacian and e) output of convolving the signal (a) with the laplacian kernel (d).

filter. We can see the diagonals again as being brighter or darker but now this effect is not an illusion, the brighter and darker diagonals are really part of the filtered image. In fact, (c) shows in blue an horizontal section at one quarter of image (b). The red curve is the section of the input image (a). We can see that the output really contains a pick in intensity on the diagonals of image (b). The plot in figure (f) shows the same for images (d) and (e).
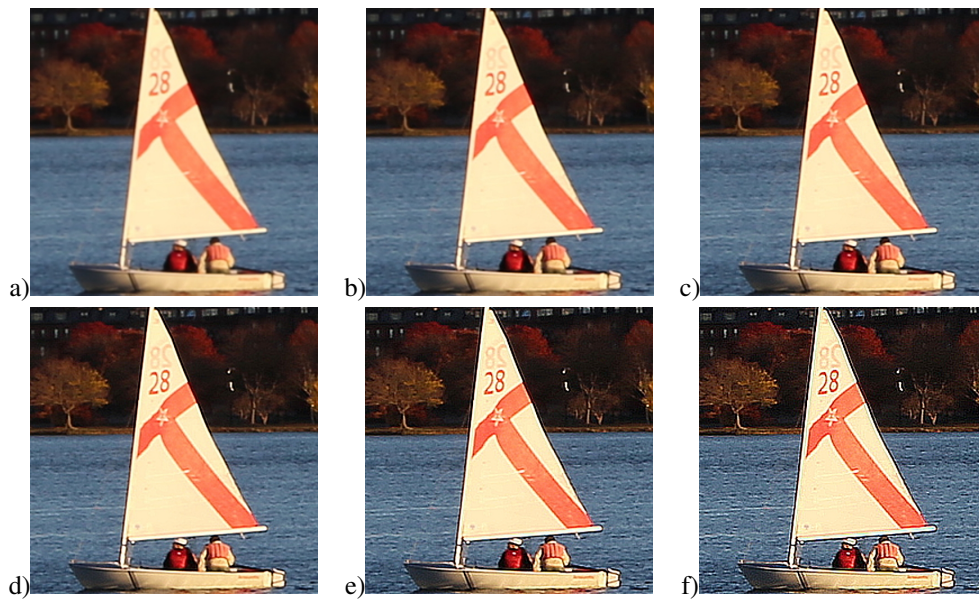
1.3.7 Sharpening filter

One example of a simple but very useful filter is a sharpening filter. The goal of a sharpening filter is to transform an image so that it appears sharper. This can be achieved by amplifying the amplitude of the high-spatial frequency content of the image. We can achieve this with a combination of filters that we have already discussed in this section.

A simple way to design a *sharpening* filter is to de-emphasize the blurry components of an image. By the linearity of the convolution operator, we're allowed to add and subtract kernels to make a new kernel that would give us the same filtered image as if we had added and subtracted the filtered outputs of each of the component kernels. For this example,

Figure 1.21
Vasarely visual illusion. Images (a) and (d) show two gray-scale pyramids. When looking at them we perceive the diagonal directions as being brighter (a) or darker (d) than their neighborhood. Images (b) and (e) show the output of the human model given by the filter from eq. 1.51. The plots (c) and (f) show, in blue, an horizontal section at one quarter of image (b) and (e) showing the intensity profile as a function of x. The red curve is the section of the corresponding input image. We can see that the output really contains a pick in intensity on the diagonals of the input image.

Figure 1.22

Sharpening achieved by subtraction of blurred components. a) Original image, b) sharpened once by filtering with kernel from eq. 1.52. Each color channel is filtered independently. c) - f) the same filter is applied successively to the previous output. In the last image the sharpening filter has been applied 5 times to the input image. The last image seems looks substantially sharper than the original image, but close inspection will reveal some artifacts.

we start with twice the original image (sharp plus blurred parts), then subtract away the blurred components of the image:

$$\text{sharpening filter} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 2 & 0 \\ 0 & 0 & 0 \end{bmatrix} - \frac{1}{16} \begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{1.52}$$

Note that the DC gain of this sharpening filter is 1. That would leave one original image in there, plus an additional component of the sharp details. The perceptual result is that of a sharpened image, Fig. 1.22. We can apply this filter successively in order to further enhance the image details. If too much sharpening is applied we might end up enhancing noise and introducing image artifacts.
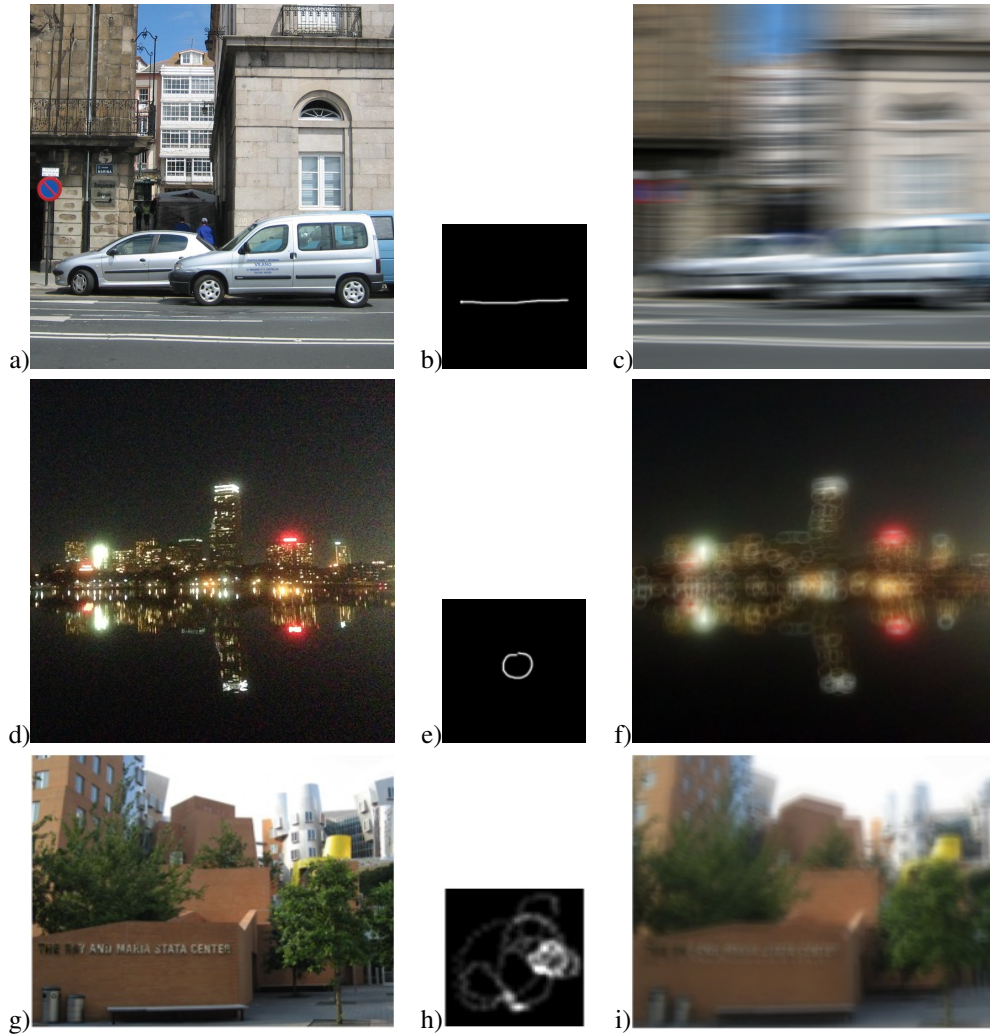
1.3.8   Motion blur and camera shake.

The Gaussian filter that we have studied can be used to model how images are blur when a picture is taken out of focus, or when we remove the eyeglasses. But there are other types of image blur that also happen in natural situations. Figure 1.23 shows examples of linear motion blur (fig 1.23.a-c), and camera shake (fig 1.23.d-i). Motion blur happens when there are objects moving in the scene or when the camera is moving while the shutter is open. Camera shake is produced due to the small camera vibration that happens while taking a picture and it is more important under low-light conditions. Camera shake can be modeled as a blur kernel, describing the camera motion during exposure, convolved with the image intensities. Depending on the shake the resulting picture looks as

Figure 1.23.d-i shows one simulated example of camera shake. In this case, the photographer is trying to capture the scene shown in fig. 1.23.d, but during exposure the camera followed a circular path. Therefore, the resulting picture (fig. 1.23.f) is the convolution of the ideal picture (fig. 1.23.d) with the convolution kernel (fig. 1.23.e). The result has probably a familiar effect for anyone that has taken pictures at night. The light sources in the picture are like delta functions that produce multiple copies of the convolution kernel. Figure 1.23.g-i shows a different example where the camera has follow a more complex trajectory.

It will be nice to have tools capable to removing the effect of camera motion to produce sharp pictures. This is an active area of research and we will discuss approaches to this problem in chapter **??**.

1.3.9   Artistic filters

Despite the simplicity of linear filters, they can be used to produce pleasing artistic effects. Combining linear filters with some simple non-linearities provides a very large number of possibilities. These types of image manipulation techniques are very popular as they can be implemented very efficiently and can be computed very fast even on mobile devices. Linear convolutional filters are the basis of applications such as Photoshop or Instagram.

Figure 1.23
a) Simulated motion blur, in this case it is produced because the picture is taken from a moving car.
d) Simulated camera shake blur (the camera has followed a circular trajectory while the shutter was open). Probably the reader has seen this type of effect in some pictures taken at night.
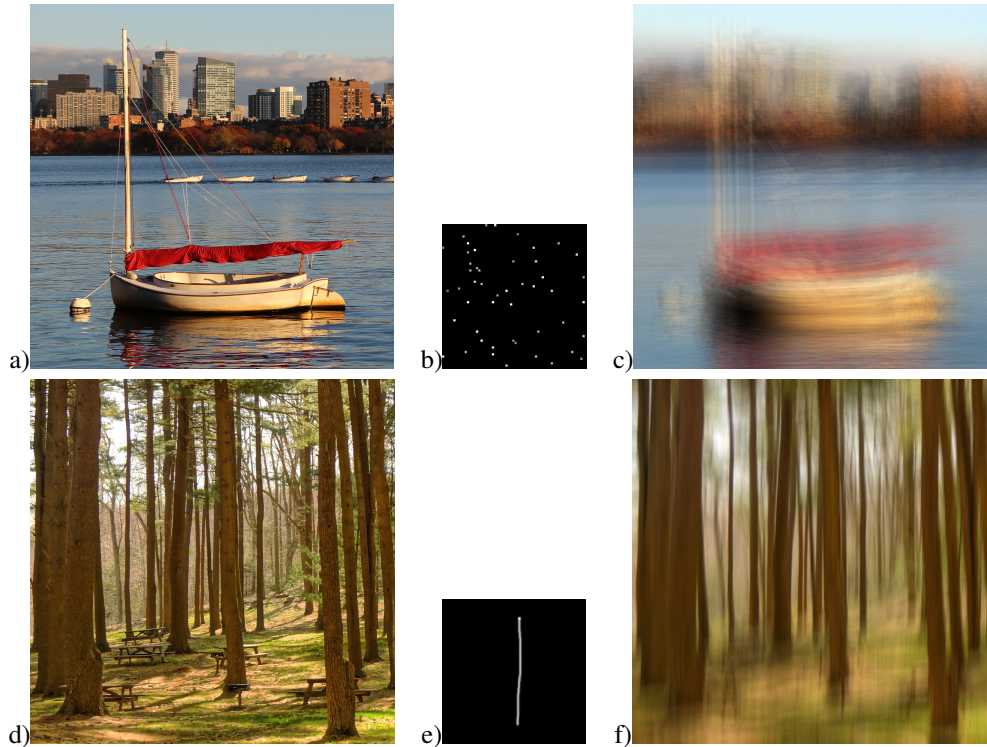
Figure 1.24
Simple linear filters can create interesting effects. a-c) Random kernel. Here, by filtering the image
(a) with a random set of impulses (b) at random locations and random positive amplitudes, creates
an image that seems to be drawn with a pencil (c). In this example, the image is 512×512 pixels and
the kernel has a support of 64×64 pixels with 50 random dots with random amplitudes between 0.5
and 1. Each color channel is filtered independently with the same kernel. d-f) Vertical motion blur.

Exploring artistic effects and playing with filters to enhance pictures is a nice way of
learning and gaining intuition about linear filter theory. Figure 1.24 shows a few linear
filters and the result they have in images. Those filters are easy to implement and will
allow you creating nice effects.

Intentional motion blur that can be achieved by moving the camera while taking a picture.
The example in figure 1.24.f simulates the effect of moving the camera up and down. The
linear filtering produces an interesting effect on the tree trunks making them become more
visually salient while attenuating other image details.

1.4   Gabor filters

Although linear filters can perform a wide range of operations, image understanding requires non-linear operators.

In the previous chapter we discussed several useful image representations: representing an image in the frequency domain and decomposing it into amplitude and phase, and also the analysis of image content across different scales and orientations. The Fourier transform was a tool that allowed us to extract that information, but only globally. For this information to be useful it needs to be localized. For instance, the analysis of orientations of local image structures can be done using image gradients (section **??**), which are localized in space. Here we will discuss Gabor filters which allow analyzing the local frequency content (amplitude and phase).

A good start for a localized image analysis is to restrict the spatial support of a sinusoidal basis function. We can multiply a Fourier basis function, $\exp(2\pi j(u_0 x + v_0 y))$ (shown here in the x direction), by a spatially localizing Gaussian window, $\exp\left(-\frac{x^2+y^2}{2\sigma^2}\right)$ to obtain a *Gabor* function, $\psi(x, y)$:

$$\psi(x, y; u_0, v_0) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \exp\left(2\pi j(u_0 x + v_0 y)\right) \tag{1.53}$$

A Gabor function is complex-valued, but we can look at the real or imaginary parts to examine cosine and sine phase local filters:

$$\psi_r(x, y; u_0, v_0) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \cos\left(2\pi j(u_0 x + v_0 y)\right) \tag{1.54}$$

$$\psi_i(x, y; u_0, v_0) = \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right) \sin\left(2\pi j(u_0 x + v_0 y)\right) \tag{1.55}$$

Figure 1.25 shows the Gaussian window and the real and imaginary parts of the Gabor function. The value of σ sets the locality of the window of analysis and the values of (u_0, v_0) adjust the orientation of the Gabor function and frequency. If the value of σ is very large, then analyzing the image with a set of Gabor functions is like the Fourier transform of the image. The Gabor filter is and oriented band pass filter. The Fourier transform of a complex Gabor function is a gaussian in the Fourier domain shifted with respect to the origin:

$$\Psi(u, v; u_0, v_0) = \exp\left(-2\pi^2\left((u - u_0)^2 + (v - v_0)^2\right)\sigma^2\right) \tag{1.56}$$

Figure 1.26 shows the magnitude of the Fourier transform of the Gabor filters.

The sine phase Gabor function is zero mean. However, the cosine phase Gabor function is not zero mean. The gaussian width σ has to be sufficiently large so that the Gabor function behaves as a zero mean filter. All the previous definitions are given in the continuous domain. The discrete version of the Gabor function is obtained by sampling the continuous functions.

a)   b)   c)

d)   e)
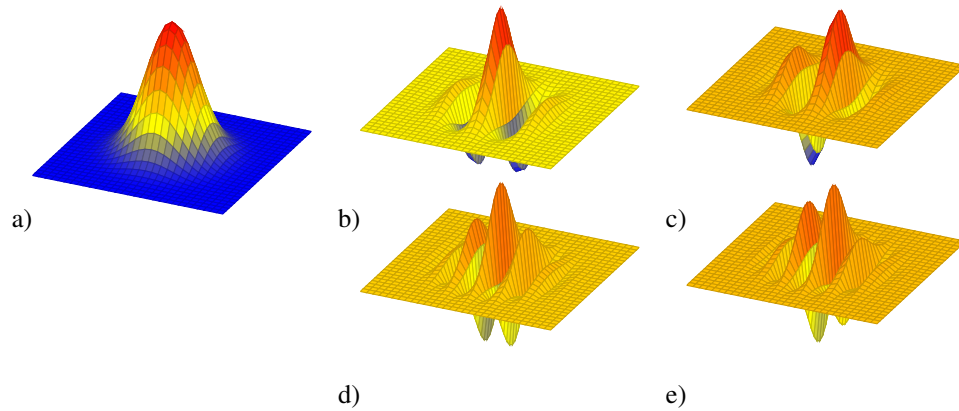
Figure 1.25
Gabor functions. a) The localizing Gaussian window ($\sigma = 1$), which can be thought of as a Gabor function for a zero frequency sinusoid. b) Cosine and c) sine phase Gabor functions with $u_0 = 0.5$ and $v_0 = 0$.
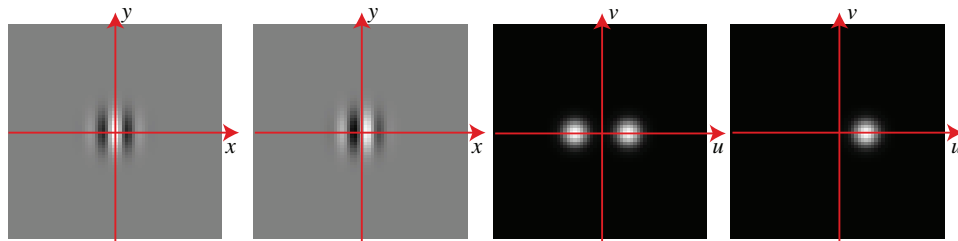
Figure 1.26
a) Cosine phase Gabor function, $\psi_r(x, y; u_0, v_0)$, b) sine phase Gabor function, $\psi_i(x, y; u_0, v_0)$, c) Fourier transform of the cosine phase Gabor function, $\Psi_r(u, v; u_0, v_0)$. d) Fourier transform of the complex Gabor function, $\Psi(u, v; u_0, v_0)$.
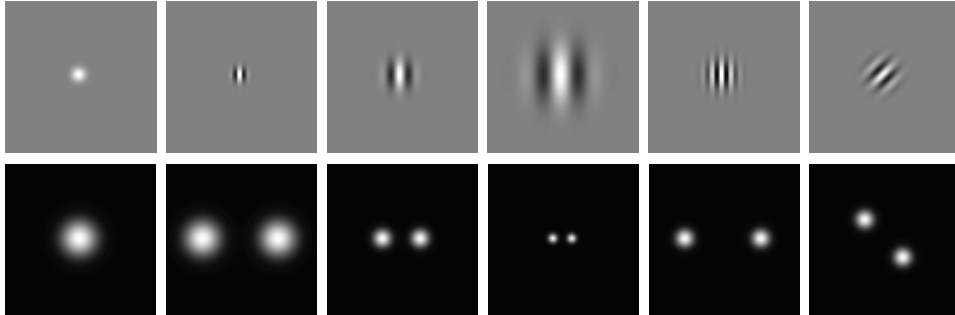
Figure 1.27
Cosine phase Gabor function tuned to different widths, frequencies, and orientations, and their corresponding Fourier transforms (only the magnitude is shown).

One important characteristic of the Gabor filters is that they are very similar to the shape of some cortical receptive fields found in the mammalian visual system **?**. This provides a hint that we're on the right track with these filters to build image representations.

The convolution of the Gabor function with an image results in a signal that depends both in space and frequency:

$$\mathbf{I}_\psi(x, y, u, v) = \mathbf{I}(x, y) \circ \psi(x, y; u, v) \tag{1.57}$$

Figure 1.28 shows the result of the convolution between a picture and the cosine and sine phase Gabor functions at three different scales ($\sigma = 2, 4, 8$). In this example, the Gabor filters are tuned to detect vertical edges. Gabor filters are useful for analyzing line or edge phase structures in images. But they have many other benefits when we combine them together in quadrature pairs.

1.4.1   Quadrature pairs and the Hilbert transform

Pairs of filters can be in a relationship to each other that is called "quadrature phase". This relationship is useful for many low-level visual processing tasks. When in quadrature phase, pairs of oriented filters can measure what is called local oriented energy, can identify contours, independently of the phase of the contour, and can measure positional changes very accurately. Let's start with the mathematical definition of quadrature.

For notational simplicity, we'll describe quadrature pair filters in the time domain, but they extend naturally to two or more dimensions, as illustrated in Figs. **??** and 1.29. Consider an even symmetric zero-mean signal, $f(t)$ and with Fourier transform $F(\omega)$, with $F(0) = 0$ because the signal is zero-mean. In the Fourier domain, two functions, $F(\omega)$ and
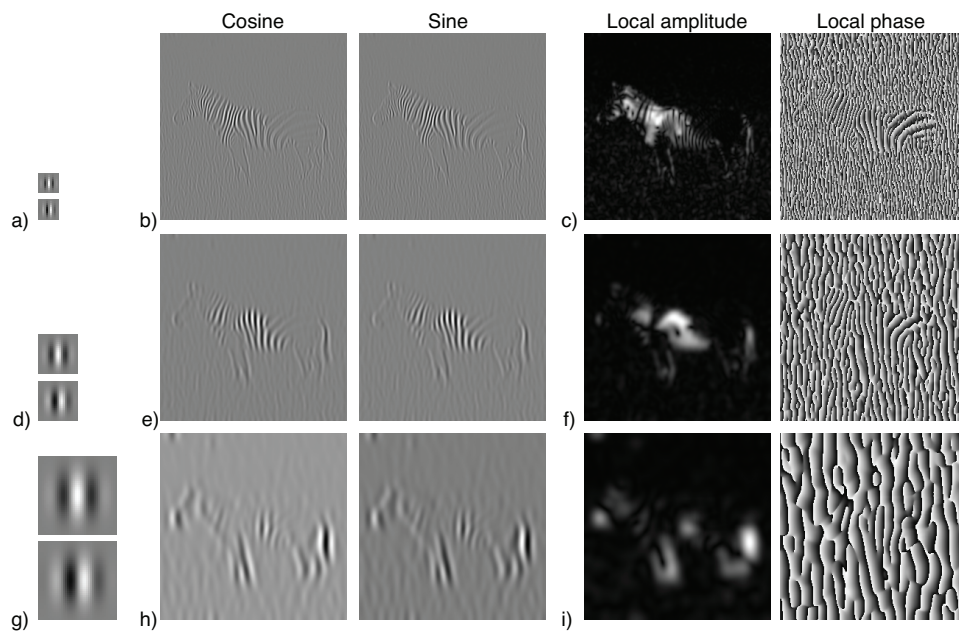
Figure 1.28
Zebra picture filtered by cosine and sine Gabor functions at three scales with $\sigma = 2, 4, 8$ and $u_0 = 1/(2\sigma)$, $v_0 = 0$. Each row shows one scale. a) Shows the cosine and sine kernels, b) cosine and sine outputs, c) magnitude and phase of the output of the complex Gabor filter.

$H(\omega)$, are said to be Hilbert transform pairs if:

$$H(\omega) = \begin{cases} -jF(\omega) & \text{if } \omega > 0 \\ jF(\omega) & \text{if } \omega < 0 \end{cases} \tag{1.58}$$

We now define the complex signal:

$$g(t) = f(t) + jh(t) \tag{1.59}$$

This signal is called *the analytic signal* and its Fourier transform has no negative frequency components:

$$G(\omega) = \begin{cases} 2F(\omega) & \text{if } \omega > 0 \\ 0 & \text{if } \omega < 0 \end{cases} \tag{1.60}$$

It is interesting to write the complex signal $g(t)$ in polar form:

$$g(t) = a(t) \exp(j\theta(t)) \tag{1.61}$$

where $a(t)$ is the instantaneous amplitude (local amplitude) and $\theta(t)$ is the instantaneous phase (local phase). This particular representation is common in communications theory, and it has been used to build image representations invariant to certain image structures as we will discuss in the following sections.

The extension of the Hilbert transform to images can be done in several ways. The most common approach in image processing is to define one direction in the frequency space n and then the Hilbert transform as follows:

$$H(\omega_x, \omega_y) = \begin{cases} -jF(\omega_x, \omega_y) & \text{if } n^T \cdot (\omega_x, \omega_y) > 0 \\ jF(\omega_x, \omega_y) & \text{if } n^T \cdot (\omega_x, \omega_y) < 0 \end{cases} \tag{1.62}$$

Two bandpass filters are said to be in quadrature if the impulse responses are Hilbert transform pairs. Sine and cosine functions of the same frequency are Hilbert transform pairs, as are sine and cosine phase Gabor functions, of the same frequency and Gaussian envelope parameters. Thus, these filter pairs are also quadrature pairs. When convolving two filters in quadrature with a signal (or image) the two outputs are also in quadrature.

The quadrature in the Gabor functions is an approximation that only holds for σ sufficiently large. For small σ the filter does not form a quadrature pair. For large σ, the real and imaginary parts of the Gabor filter (cosine and sine phase local filters) are filters in quadrature with the vector $n = (u_0, v_0)$ pointing in the direction of the central frequency of the Gabor function.

1.4.2   Local amplitude

Let $f(x, y)$ and $h(x, y)$ be bandpass filters in quadrature, and let $I_f(x, y)$ and $I_h(x, y)$ be the result of convolving the signal $I(t)$ with $f(x, y)$ and $h(x, y)$, respectively. Then, the squared local amplitude is a measure of the image power within the passband of the filters in the
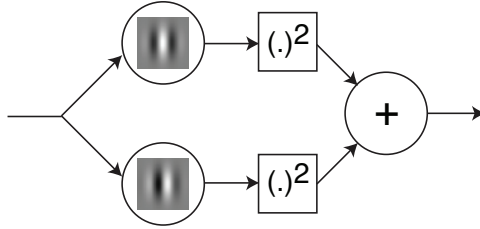
Figure 1.29
Computation of localized amplitude. The input is filtered by a pair of quadrature Gabor filters. Each filter output is squared and the result is added.

local neighborhood of (x, y):

$$a^2(x, y) = I_f(x, y)^2 + I_h(x, y)^2 \tag{1.63}$$

Figure 1.29 shows the steps to compute the local amplitude of an input image using Gabor filters.

To see how useful the local amplitude is, let's start by computing it for some simple images. If the input image is a delta, $\mathbf{I}(x, y) = \delta(x, y)$, and the filters f and h are the cosine and sine phase Gabor filters, then, the local amplitude image is:

$$\begin{aligned}
a(x, y) &= \sqrt{I_f(x, y)^2 + I_h(x, y)^2} = \\
&= \sqrt{\psi_r^2(x, y; u_0, v_0) + \psi_i^2(x, y; u_0, v_0)} = \\
&= \frac{1}{2\pi\sigma^2} \exp\left(-\frac{x^2 + y^2}{2\sigma^2}\right)
\end{aligned} \tag{1.64}$$

The amplitude of the delta function is the gaussian envelope of the Gabor function. This result is independent of the contrast of the input image. For instance, if the input is $\mathbf{I}(x, y) = -\delta(x, y)$, the amplitude image does not change. Figure 1.30.a shows an image with two impulses of opposite signs. Figures 1.30.b show the output of the cosine and phase filters, and figure 1.30.c shows the local amplitude $a(x, y)$. The local amplitude is two Gaussians centered on each impulse.

This contrast invariance property is specially useful when using the local amplitude to localize edges in images. Figures 1.30.d shows an image composed of several squares. Each square is defined by different polarities with respect to the background. Two of the squares are solid while the other two are only defined by lines. The local amplitude, Figures 1.30.f, provides a detector for the square boundaries that is invariant to all those changes. The differences between all the squares are encoded in the local phase image as we will discuss in the following section. Figure 1.28 shows the local amplitude signal computed on real images.
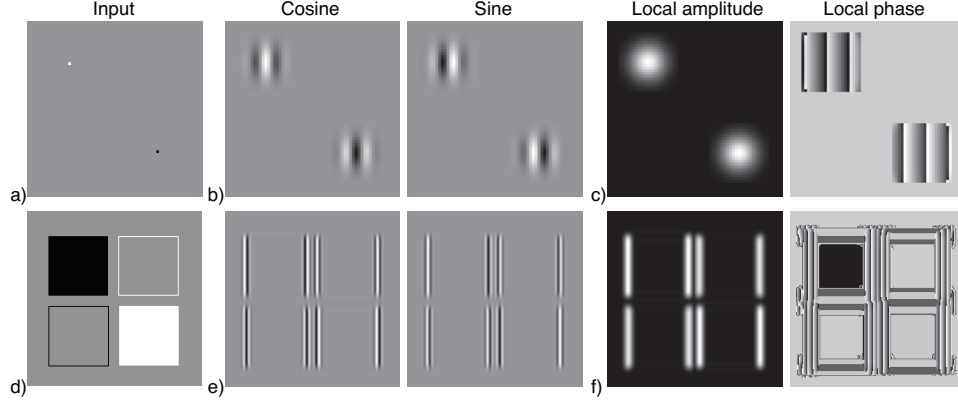
Figure 1.30

Examples of Gabor outputs to illustrate the contrast invariances present in the local amplitude. In these examples the Gabor filters are centered along the horizontal frequency axis ($v_0 = 0$) therefore detecting only vertical edges.

One property to notice in all these examples, is that, although the images $I_f(x, y)$ and $I_h(x, y)$ are band-pass, the amplitude $a^2(x, y)$ is a low-pass image. To check this it is interesting to compute the Fourier transform of $a^2(x, y)$.

Consider the bandpass filter, $f(x, y)$, with Fourier transform F(u, v). We can write F(u, v) as the sum of two functions F$^+$(u, v) and F$^-$(u, v) so that F$^+$(u, v) = 0 for $n^T \cdot (\omega_x, \omega_y) < 0$ and F$^-$(u, v) = 0 for $n^T \cdot (\omega_x, \omega_y) > 0$. If $f(x, y)$ is even symmetric then we have that F$^+$(u, v) = F$^-$($-u, -v$), for instance the cosine phase Gabor filter. This decomposition is illustrated in figure 1.31.a. If the Fourier transform of $I_f(x, y)$ is I$_f(u, v)$, then the Fourier transform of $I_f^2(x, y)$ is I$_f(u, v) \circ$ I$_f(u, v)$. Then the Fourier transform of $I_f^2(x, y)$ is:

$$\text{I}_f \circ \text{I}_f = \text{I}(\text{F}^+ + \text{F}^-) \circ \text{I}(\text{F}^+ + \text{F}^-) = \text{IF}^+ \circ \text{IF}^+ + \text{IF}^- \circ \text{IF}^- + 2\text{IF}^+ \circ \text{IF}^- \qquad (1.65)$$

If $h(x, y)$ is the Hilbert transform of $f(x, y)$, then H(u, v) = j(F$^+$(u, v) − F$^-$(u, v)), as illustrated in figure 1.31.b. The Fourier transform of $I_h^2(x, y)$ is:
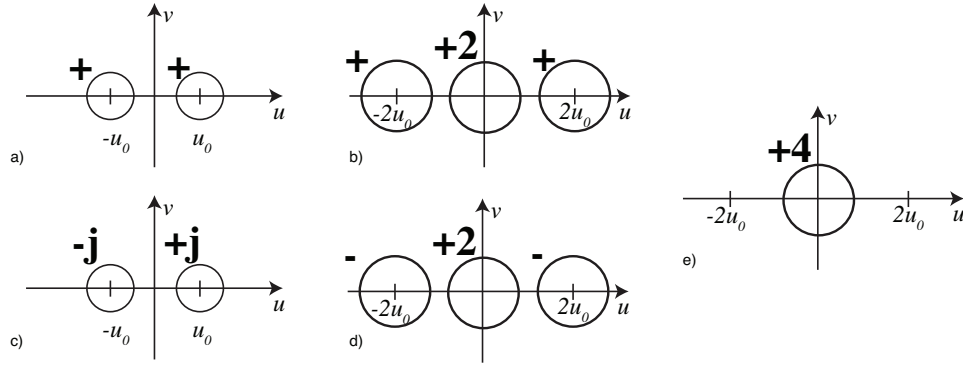
$$\text{I}_h \circ \text{I}_h = j\text{I}(\text{F}^+ - \text{F}^-) \circ j\text{I}(\text{F}^+ - \text{F}^-) = -\text{IF}^+ \circ \text{IF}^+ - \text{IF}^- \circ \text{IF}^- + 2\text{IF}^+ \circ \text{IF}^- \qquad (1.66)$$

Note that if IF$^+$ is centered around the frequency (u_0, v_0), then (IF) $\circ$ (IF) is centered around ($2u_0, 2v_0$). Also, (IF$^+$) $\circ$ (IF$^-$) is centered around the frequency $(0, 0)$. This is shown in figures 1.31.b and d.

The Fourier transform of the local amplitude $a^2(x, y) = I_f^2 + I_h^2$ is:

$$\text{I}_f \circ \text{I}_f + \text{I}_h \circ \text{I}_h = 4\text{IF}^+ \circ \text{IF}^- \qquad (1.67)$$

Eq. (1.67) reveals that $a^2(x, y)$ is a low-pass signal, as shown shown in figure 1.31.e.

Figure 1.31

Sketch of the derivation of the Fourier transform of the local amplitude. a) frequency response of even filter f, b) a) frequency response of odd filter h, c) Fourier transform of f^2, d) Fourier transform of h^2, e) Fourier transform of $a^2 = f^2 + h^2$

1.4.3   Local phase

The local phase is

$$\theta(x, y) = \angle \left[I_f(x, y) + j I_h(x, y) \right] \tag{1.68}$$

where $\theta(x, y)$ is the instantaneous phase (local phase).

For oriented, spatial filters, cycling through the phase of the quadrature pair of filters can generate motion along the direction of the phase change.

1.4.4   Gabor filter bank

As shown in fig. 1.26, 2D Gabor filters are selective in spatial frequency. It is very useful to work with sets of Gabor filters, each selective to a different spatial frequency. Figure 1.32 shows two different arrangements of Gabor filters. Figure 1.32.a shows a set of Gabor filters sampling the frequency domain using a rectangular grid. Figure 1.32.c shows the corresponding (cosine) Gabor kernels. All the functions have the same σ. Figure 1.32.b shows a polar arrangement of Gabor functions, and Figure 1.32.d the spatial kernels. Here, the Gaussian width σ is proportional to the distance between the central frequency and the origin. This produces filters that are rotated and scaled versions of each other.

If we write in detail the convolution from eq. 1.57 we can see that the convolution with a Gabor function is like doing a Fourier transform of the image after applying to it a Gaussian window:

$$f(x, y, u, v) = \iint \mathbf{I}(x', y') \, \psi(x - x', y - y'; u, v) \, dx' dy' =$$

$$= \iint \mathbf{I}(x', y') \, g(x - x', y - y') \exp\left(2\pi j \left(u(x - x') + v(y - y')\right)\right) dx' dy' =$$
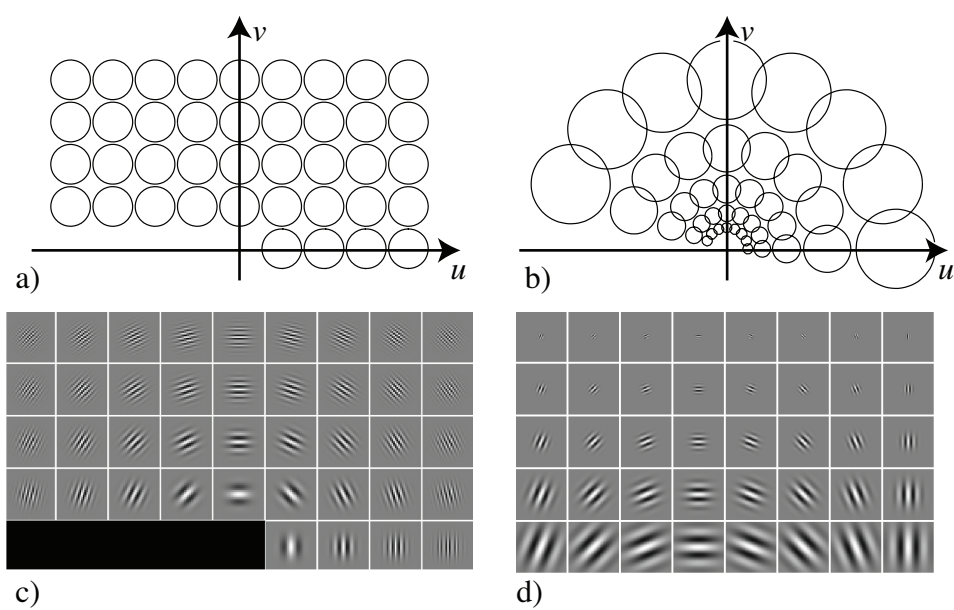
Figure 1.32
Examples of Gabor sets.

if we extract the term that does not depend on (x', y') and we also use the symmetry of the Gaussian window, we get

$$= \exp\left(2\pi j\left(ux + vy\right)\right) \iint \mathbf{I}\left(x', y'\right) g(x' - x, y' - y) \exp\left(-2\pi j\left(ux' + vy'\right)\right) dx'dy' \quad (1.69)$$

The analogous to the Fourier transform is obtained when we vary the central frequency of the Gabor function (u, v). The Gabor transform is not self inverting.

1.5   Steerable filters and orientation analysis

One question that arises with oriented filters is how to change their orientation. More precisely, what we want is, given a filter $f(x, y)$, we want to transform it into a continuous function $f(x, y, \theta)$ of angle θ. The angle θ specifies the rotation of the original filter $f(x, y)$.

In the case of the Gabor filters (eq. 1.53), each orientation requires convolving the image with a Gabor function tuned to that orientation. But do we need to create a new Gabor function for each orientation, or can we interpolate between a fixed number of predefined oriented filter outputs? How many orientations need to be sampled?

We'd like an analog in orientation for the Nyquist sampling theorem in space: given a certain number of discrete samples in orientation (or space), can one interpolate between the samples and synthesize what would have been found from having a filter (or a spatial sample) at some arbitrary, intermediate orientation? The answer is yes, and the number of filter samples needed to interpolate depends on the form of the filter.

From section **??**, we described the simplest example of an oriented filer: a gaussian derivative. As we discussed in eq. **??**, we can synthesize a directional derivative in any direction as a linear combination of derivatives in the horizontal and vertical directions. By the linearity of convolution, that applies to the derivative applied to any filter or image, as well. It can be seen that the "steering equation" for the first derivative of a Gaussian filter is

$$g_x(x, y, \theta) = \cos(\theta)g_x(x, y) + \sin(\theta)g_y(x, y) \quad (1.70)$$

where g_x and g_y are the Gaussian derivatives along x and y, and $g_x(x, y, \theta)$ is the derivative along the direction defined by the angle θ. The interpolation functions are $k_1(\theta) = \cos(\theta)$ and $k_2(\theta) = \sin(\theta)$.

In fact, all higher order Gaussian derivatives have the same property but the number of basis filters changes. For instance, for second order Gaussian derivatives, the steering equation is:

$$\begin{aligned} g_{x,x}(x, y, \theta) &= g_{xx}(\cos(\theta)x - \sin(\theta)y, \sin(\theta)x + \cos(\theta)y) = \\ &= \cos^2(\theta)g_{xx}(x, y) + \sin^2(\theta)g_{yy}(x, y) - 2\cos(\theta)\sin(\theta)g_{xy}(x, y) \end{aligned} \quad (1.71)$$

To interpolate the derivative along any orientation requires three basis filters and the interpolation functions are: $k_1(\theta) = \cos^2(\theta)$, $k_2(\theta) = \sin^2(\theta)$, and $k_3(\theta) = -2\cos(\theta)\sin(\theta)$. The
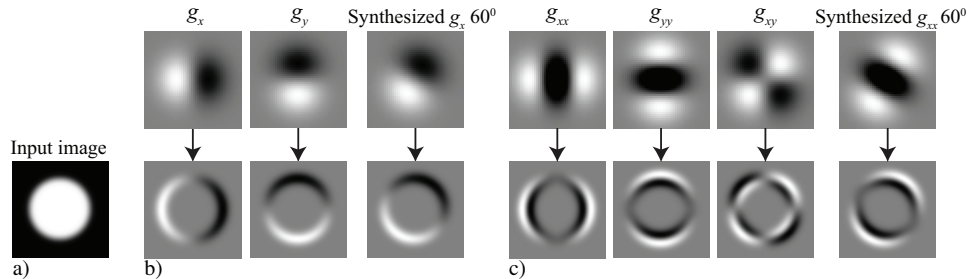
Figure 1.33

The simplest steerable filters: a 1st order derivative filter of any orientation can be synthesized from a linear combination of two basis filter derivatives. The 2nd order derivative needs three basis.
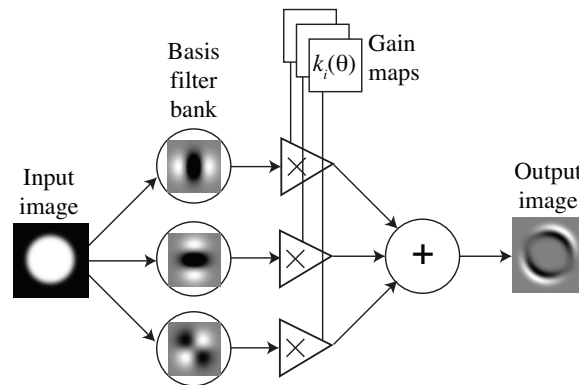
Figure 1.34

Architecture for steerable filters.

minus sign in k_3 is due to the positive direction of θ to be counter-clockwise. Figure 1.33 shows examples for the 1st and 2nd order Gaussian derivatives.

This leads to an architecture for processing images with oriented *steerable* filters shown in figure 1.34. The input images pass through a set of "basis filters", then the outputs of those filters are modulated with a set of "gain maps" (which can be different at each pixel). Those gain maps adjust the linear combinations of the basis filters to allow the input filter to be steered to the desired orientations at each position.

In the case of oriented Gabor filters it is not possible to reconstruct exactly any filter orientation by interpolating filter responses. It is interesting to study the conditions in which interpolation gives exact results.

1.5.1   Steering theorem

Let's make the previous observation more precise and general. How many basis filters does it take to steer any given filter? You could imagine that will depend on how sharply oriented the filter is. A circularly symmetric filter takes just one basis function to synthesize all other orientation responses, and a very narrow filter will take quite a few. This is quantified by steering theorems.

Lets consider a filter with impulse response $f(x, y)$. For convenience, it is better to write the filter response in polar coordinates, $f(r, \phi)$ The "steering condition" is the requirement that the rotated filter, $f(r, \phi - \theta)$, be a linear combination of a set of basis filters which are rotated versions of itself, $f(r, \phi - \theta_m)$ with $m \in (1, M)$. The steering condition is:

$$f(r, \phi - \theta) = \sum_{m=1}^{M} k_m(\theta) f(r, \phi - \theta_m). \tag{1.72}$$

If we express the filter to be steered as a Fourier series in angle (using complex exponentials for notational convenience), we have

$$f(r, \phi) = \sum_{n=-N}^{N} a_n(r) \exp(jn\phi) \tag{1.73}$$

Substituting Eq. (1.73) into Eq. (1.72), we have an equation for the interpolation functions, $k_j(\theta)$. The steering condition, Eq. (1.72), holds for functions expandable in the form of Eq. (1.73) if and only if the interpolation functions $k_j(\theta)$ are solutions of:

$$\begin{pmatrix} 1 \\ \exp(j\theta) \\ \dots \\ \exp(jN\theta) \end{pmatrix} = \begin{pmatrix} 1 & 1 & \dots & 1 \\ \exp(j\theta_1) & \exp(j\theta_2) & \dots & \exp(j\theta_M) \\ \vdots & \vdots & \vdots & \vdots \\ \exp(jN\theta_1) & \exp(jN\theta_2) & \dots & \exp(jN\theta_M) \end{pmatrix} \begin{pmatrix} k_1(\theta) \\ k_2(\theta) \\ \vdots \\ k_M(\theta) \end{pmatrix}. \tag{1.74}$$

Let's check this for a simple example. Our derivative of a gaussian filter (eq. **??**) is x times a Gaussian. When changing to polar coordinates $x = r\cos(\theta)$, which gives an angular distribution times a radially symmetric Gaussian, $g(r)$, when written in polar coordinates. This requires two complex exponentials to write (to create the $\cos(\theta)$ from complex exponentials) and thus requires two basis functions to steer.

Sometimes its more convenient to think of the filters as polynomials times radially symmetric window functions (this is the case for high-order Gaussian derivatives). Then you can show Freeman and Adelson (1991) that for an Nth order polynomial with even or odd symmetry N+1 basis functions are sufficient.

Although everything has been derived in the continuous domain, steerability is a property that still holds after sampling the filter function. This is because spatial sampling and
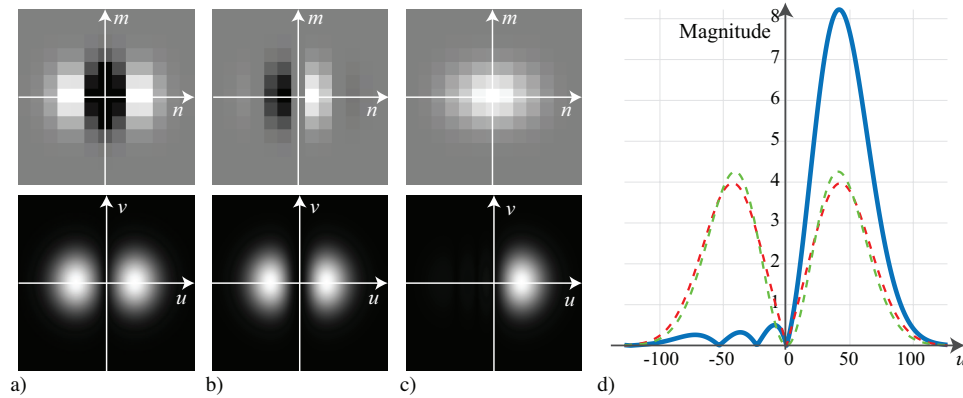
Figure 1.35

Quadrature pair, $g_{xx}[n, m]$ and $h_{xx}[n, m]$, and its Fourier transform. The filters are sampled in space and cropped into a window of 13×13 pixels. The DFT is computed by zero padding to 256×256 pixels. a) g_{xx} is even phase; b) h_{xx} is odd phase. c) The sum of their squares reveals the square of their Gaussian envelopes. d) The plot shows a section of the magnitude of their Fourier transforms for $v = 0$. h_{xx} is a sampled 3rd order polynomial approximation to the Hilbert transform of g_{xx}, so their power spectra may not be exactly the same). In blue it is shown the magnitude of the analytic filter $g_{xx} + jh_{xx}$. It has double amplitude, and the content for negative frequencies is close to zero.

steerability are interchangeable: the weighted sum of spatially sampled function is equal to the spatial sampling of the same weighted sum of continuous basis functions.

For computational efficiency, it's more convenient to have the basis filters all be x-y separable functions. In many cases, it's straightforward to find such basis functions, and where it's not, there are simple numerical methods to find the best fitting x-y separable basis set. See for example **?**.

1.5.2   Steerable quadrature pairs

As in the case of Gabor filters, it is useful to build quadrature pairs of steerable filters. Steerable-quadrature filters allow for arbitrary shifts both in orientation and in phase. We can design such filters. For instance, let's consider the second order derivatives of a Gaussian with $\sigma^2 = 1/2$ and normalized so that the integral over all the space of its squared magnitude equals 1:

$$
\begin{aligned}
g_{xx}(x, y) &= 0.9213(2x^2 - 1) \exp\left(-(x^2 + y^2)\right) \\
g_{xy}(x, y) &= 1.843(xy) \exp\left(-(x^2 + y^2)\right) \\
g_{yy}(x, y) &= 0.9213(2y^2 - 1) \exp\left(-(x^2 + y^2)\right)
\end{aligned}
\tag{1.75}
$$
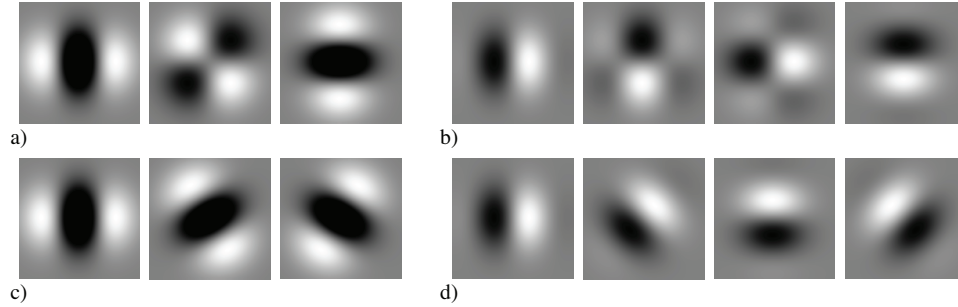
Figure 1.36
(a) Second derivative of Gaussian, x-y separable steerable basis set. c) Non-separable basis spanning the same space as the filters of (a). b) Approximation to Hilbert transform of second derivative of Gaussian, x-y steerable basis set. (d) Non-separable basis set spanning the same space as the filters of (b).

It is possible to get a good approximation to its Hilbert transform using a Gaussian times a third order odd polynomial. The approximation to the Hilbert transform of $g_{xx}(x, y)$ is:

$$h_{xx}(x, y) = -0.9780(-2.254x + x^3) \exp\left(-(x^2 + y^2)\right) \tag{1.76}$$

$g_{xx}(x, y)$ and its Hilbert transform $h_{xx}(x, y)$ have the same spectral content, but the opposite phase. Figure. 1.35 analyzes the quality of the approximation. The figure shows the quadrature pair, g_{xx} and h_{xx}, sampled in space and cropped into a window of 13×13 pixels.

To steer $h_{xx}(x, y)$ to an angle θ, this approximation requires 4 basis functions, and not just 3 as for the 2nd order derivative of a Gaussian. The other three functions needed are:

$$\begin{aligned}
h_2(x, y) &= -0.9780(-0.7515 + x^2)y \exp\left(-(x^2 + y^2)\right) \\
h_3(x, y) &= -0.9780(-0.7515 + y^2)x \exp\left(-(x^2 + y^2)\right) \\
h_4(x, y) &= -0.9780(-2.254y + y^3) \exp\left(-(x^2 + y^2)\right)
\end{aligned} \tag{1.77}$$

These functions have been optimized in order to be x-y separable. The basis function for steerability are not unique. For instance, figure 1.36.a and 1.36.c show two basis functions that span all rotations of g_{xx}. Figure 1.36.a has separable filters corresponding to g_{xx}, g_{xy} and g_{yy}, and figure 1.36.c shows three rotated versions of g_{xx} at 0, 60 and 120 degrees. Figure 1.36.b and 1.36.d show two basis functions that span all rotations on h_{xx}. Spatial scaling of the filters will result in changing σ.

Putting it all together, can we compute oriented energy as a function of angle, for all angles, just from the 7 basis filter responses shown in Figure 1.36.

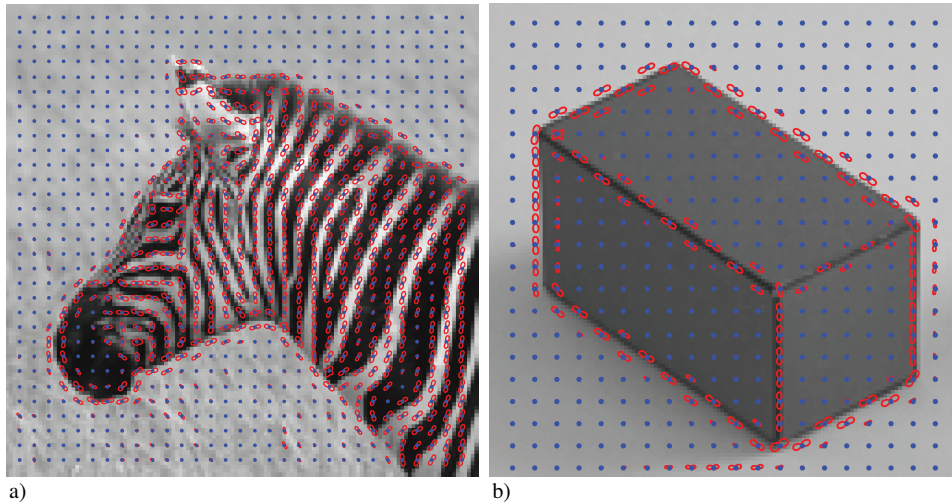$$E(x, y, \theta) = g_{xx}(x, y, \theta)^2 + h_{xx}(x, y, \theta)^2 \tag{1.78}$$
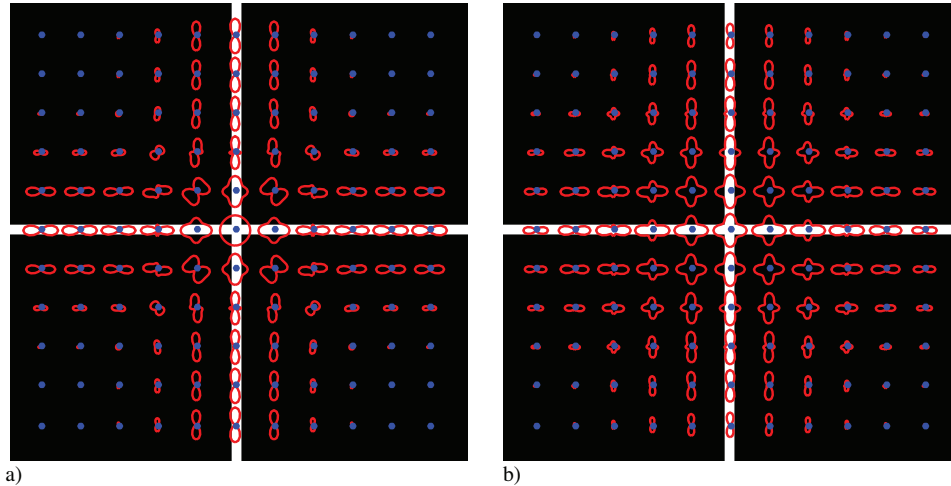
Figure 1.37

(Polar plots of orientation energy as a function of angle, computed using g_{xx}, h_{xx} filters. a) Note the non-superposition of orientated energies near the junction of the two lines. b) Spatially blurring the oriented energy components of the filters results in much improved linear superposition of the orientation plots, removing spurious interference terms, as described in the text.

From the basis filter responses we can form polar plots of the oriented energy as a function of angle, Fig. 1.37. Note some strange goings on at intersections using the g_{xx}, h_{xx} filters. You might think this was a result of simply not enough angular resolution from those filters, and indeed using 4th order gaussian derivatives and its quadrature pair doesn't suffer from that problem. But actually the g_{xx}, h_{xx} filters do have enough angular resolution, and the issue is a more subtle one.

When there are two oriented structures within the passband of the quadrature pair filters, the sum of the energies of the individual structures is not the same as the energy of the sum of the structures. Because we're squaring to find the energies, the combination of multiple structures isn't linear. As the figure 1.37.a shows, when there are two oriented structures within the passband, when the filter responses are squared, the convolution in the Fourier domain picks up extra cross-terms from the one oriented structure interacting with the other, in addition to the desired term from simply squaring all the frequency responses individually within the passband. These cross terms show up as spurious spatial frequencies in the energy term, and we can get rid of them by spatially low-pass filtering the squared oriented energy responses. Using the blurred squared basis filter responses, we get much cleaner oriented energy as a function of angle plots, even with the g_{xx}, h_{xx} filters in the junctions, Fig. 1.37.b.

Fig. 1.38 shows two examples on real images.

a)   b)

Figure 1.38
Polar plots of orientation energy as a function of angle, computed using g_{xx}, h_{xx} filters in two images.

We can also make steerable filters in 3 dimensions, allowing us to analyze medical volumetric data, or to spatio-temporal volumes to measure image motion.

1.6   Non-linear spatial filters

In this section we have exclusively focused on linear filters. Non-linear filters are hard to characterize and the number of possibilities is endless. But there are a number of non-linear filters that are closely related to some of the linear filters we have seen before and it is worth studying them here.

1.6.1   Bilateral filtering

- bilateral filtering: related to gaussian filtering
  - do small perturbation analysis

1.6.2   Local contrast enhancement

- local contrast enhancement

1.7   Conclusion

In this chapter we have talked about:

- Signals, images
- Representations in the Space and frequency domain

Important topics that we have not described:

- We have not gone in detail in recursive filters. Maybe we should add another example in the temporal domain?
- We have not described how to connect difference equations and the transfer function of a system.
- We have not described how to analyze the stability of recursive systems.
- Feedback and feedforward systems

Bibliography

Freeman, W. T., and E. H. Adelson. 1991. The design and use of steerable filters. *IEEE Pat. Anal. Mach. Intell.* 13 (9): 891–906.